

Algorytmy grafowe (AGR 320)

semestr letni 2004/2005

Michał Karoński

Wydział Matematyki i Informatyki UAM

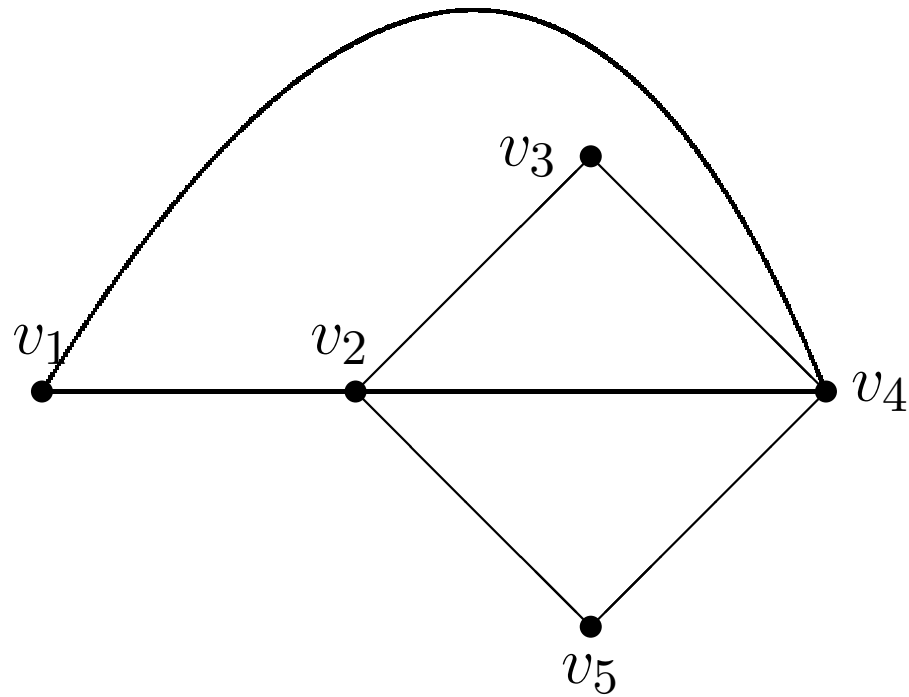
1. Rodzaje grafów

- grafy proste
- multigrafy
- grafy skierowane (digrafy)
- grafy ważone
- hipergrafy

2. Reprezentacje grafów

2.1 Macierz przyległości (sąsiedztwa)

Definicja (Macierz przyległości). Dany jest graf $G = (V, E)$, przy czym $V = \{v_1, v_2, \dots, v_n\}$ jest zbiorem wierzchołków, to $n \times n$ macierz $A(G) = (a_{ij})$, gdzie a_{ij} jest liczbą krawędzi łączących wierzchołki v_i oraz v_j nazywa się macierzą przyległości grafu G . W przypadku grafu skierowanego a_{ij} jest liczbą łuków z wierzchołka v_i do v_j .



Rysunek 1: Przykład grafu prostego $G = (V, E)$, gdzie $V = \{v_1, v_2, v_3, v_4, v_5\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$, $e_1 = v_1v_2$, $e_2 = v_2v_3$, $e_3 = v_2v_5$, $e_4 = v_3v_4$, $e_5 = v_2v_4$, $e_6 = v_4v_5$, $e_7 = v_1v_4$.

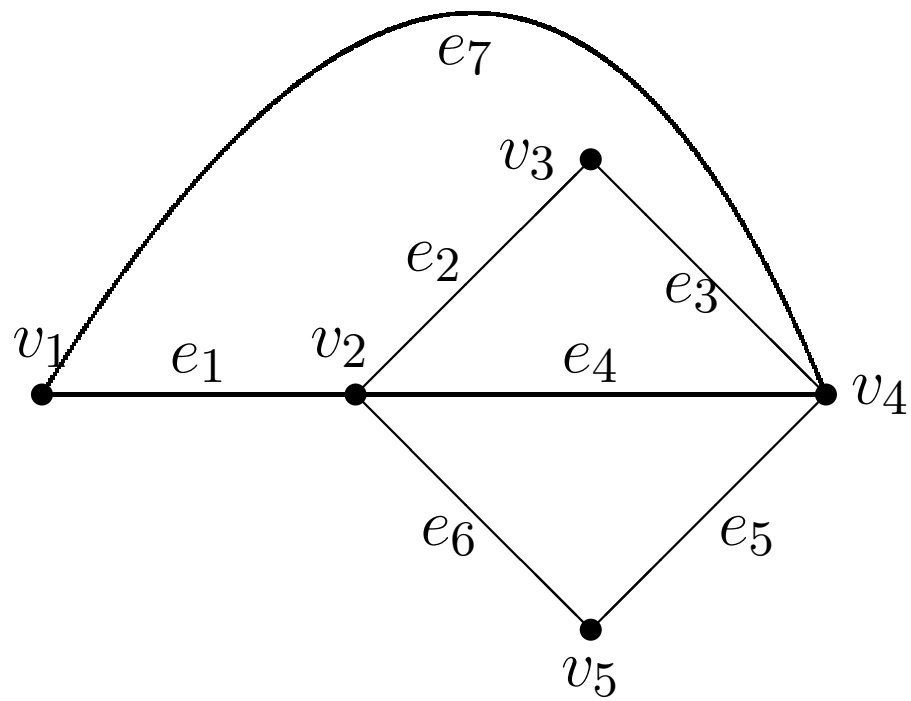
Przykład . *Macierz przyległości dla grafu przedstawionego na Rysunku 1.*

$$A(G) = \begin{matrix} & \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{matrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} .$$

- macierz przyległości (sąsiedztwa) $A(G)$ jest macierzą binarną (dla grafu prostego)
- macierz przyległości wymaga $|V|^2 = n^2$ bitów pamięci
- jeżeli w jest długością słowa maszynowego, to każdy wiersz macierzy przyległości można zapisać jako ciąg n bitów w $\lceil n/w \rceil$ słowach maszynowych ($\lceil x \rceil$ oznacza najmniejszą liczbę całkowitą nie mniejszą niż x)
- graf prosty: $A(G)$ jest symetryczna - $n(n - 1)/2$ bitów
- graf skierowany: $n \lceil n/w \rceil$
- reprezentacja macierzowa jest korzystniejsza dla grafów **gęstych**

2.2 Macierz incydencji

Definicja (Macierz incydencji). Dany jest graf $G = (V, E)$, przy czym $V = \{v_1, v_2, \dots, v_n\}$ jest zbiorem wierzchołków natomiast $E = \{e_1, e_2, \dots, e_m\}$ zbiorem krawędzie grafu, to macierz $M(G) = (m_{ij})$, $1 \leq i \leq n$, $1 \leq j \leq m$, gdzie liczba $m_{ij} \in \{0, 1, 2\}$ oznacza ile razy v_i oraz e_j są incydentne (2 występuje w przypadku pętli), jest macierzą incydencji grafu G .



Przykład . *Macierz incydencji dla grafu przedstawionego na Rysunku 1.*

$G = (V, E)$: $|V| = 5$, $|E| = 7$, $V = \{v_1, v_2, v_3, v_4, v_5\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$, $e_1 = v_1v_2$, $e_2 = v_2v_3$, $e_3 = v_2v_5$, $e_4 = v_3v_4$, $e_5 = v_2v_4$, $e_6 = v_4v_5$, $e_7 = v_1v_4$.

$$M(G) = \begin{matrix} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{matrix}$$

- Macierz incydencji wymaga $|V||E|$ bitów pamięci, co może być liczbą większą niż $|V|^2$ bitów zajmowanych przez macierz przyległości, ponieważ liczba krawędzi $|E|$ jest często większa niż liczba wierzchołków $|V|$.
- W niektórych jednak przypadkach może być korzystniejsze użycie macierzy incydencji, niż macierzy przyległości pomimo zwiększonej zajętości pamięci.
- Macierze incydencji są szczególnie dogodne przy rozpatrywaniu obwodów elektrycznych i układów przełączających.

2.3 Lista krawędzi

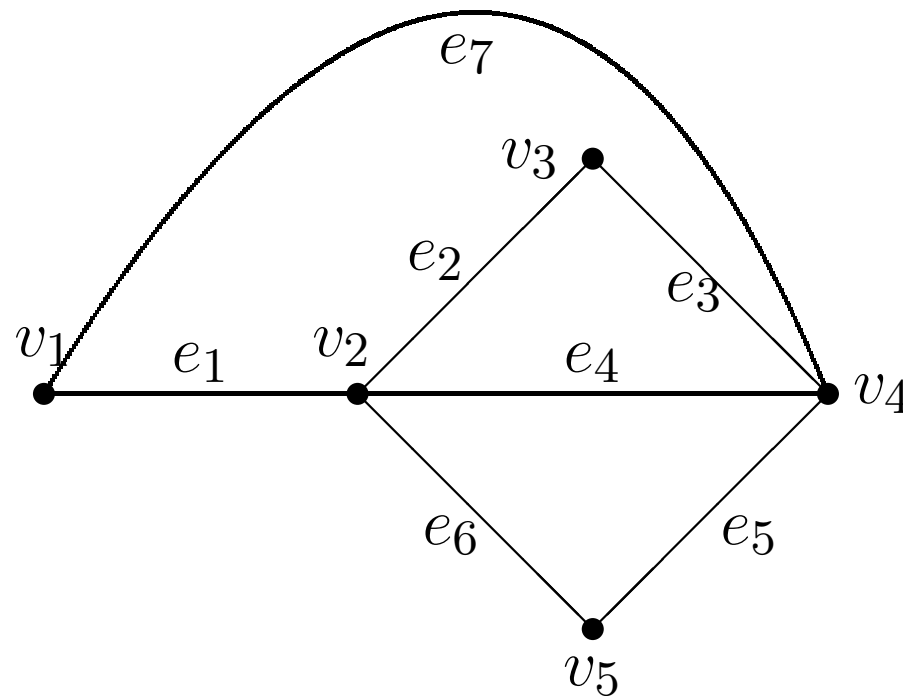
- Innym, często stosowanym sposobem reprezentacji grafu jest wypisanie wszystkich jego krawędzi jako par wierzchołków. Tak na przykład, graf z Rysunku 1 byłby przedstawiony jako lista następujących nieuporządkowanych par:
 $(v_1, v_2), \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_5\}.$
- Dla grafu skierowanego, były by to uporządkowane pary wierzchołków odpowiadające łukom.

- liczba bitów potrzebna do zaetykietowania (etykietami od 1 do n) wierzchołków grafu G jest równa b , gdzie $2^{b-1} < n \leq 2^b$, czyli $b = \lfloor \log_2 n \rfloor + 1$
- całkowita zajętość pamięci jest równa $2|E|b$ bitów
- ten sposób reprezentacji jest bardziej ekonomiczny niż macierz przyległości, jeżeli $2|E|b < |V|^2$
- reprezentacja “listowa” jest korzystniejsza dla grafów **rzadkich**
- przechowywanie i przekształcanie grafu w komputerze jest trudniejsze (na przykład przy badaniu spójności grafu)

2.4 Dwie tablice liniowe

- modyfikacją listy krawędzi - przedstawienie grafu za pomocą dwóch tablic liniowych:
$$F = (f_1, f_2, \dots, f_e), \quad H = (h_1, h_2, \dots, h_e).$$
- elementy tablic: etykiety wierzchołków, $e = |E|$
- G graf skierowany: i -ty łuk, e_i , prowadzi od wierzchołka f_i , do wierzchołka h_i
- G graf nieskierowany: krawędź e_i łączy f_i i h_i .
- dogodna reprezentacja do sortowania w grafach ważonych

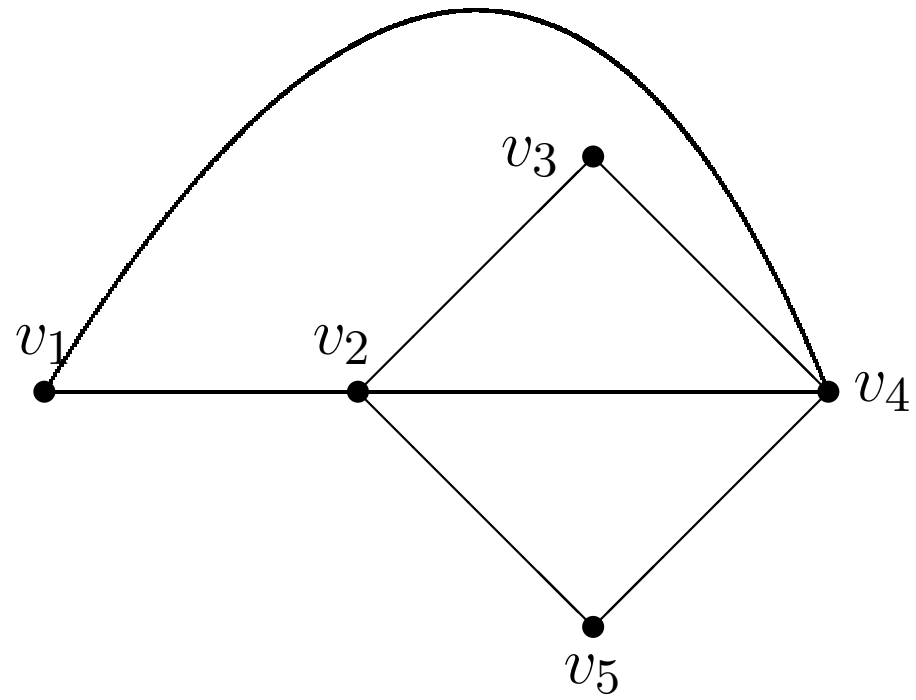
Przykład . Dwie tablice liniowe dla grafu przedstawionego na Rysunku 1.



$$F = (v_1, v_2, v_2, v_3, v_2, v_4, v_1), \quad H = (v_2, v_3, v_5, v_4, v_4, v_5, v_4).$$

2.5 Lista wierzchołków sąsiednich (następników)

- efektywna metoda reprezentacji grafów stosowana w przypadku gdy stosunek $|E|/|V|$ nie jest duży
- dla każdego wierzchołka v tworzymy listę (tablicę), której pierwszym elementem jest v ; a pozostałymi elementami są:
 - wierzchołki będące sąsiadami wierzchołka v (w przypadku grafu nieskierowanego)
 - bezpośredni następnicy wierzchołka v , tzn. wierzchołki, do których istnieje łuk z wierzchołka v (w przypadku grafu skierowanego)



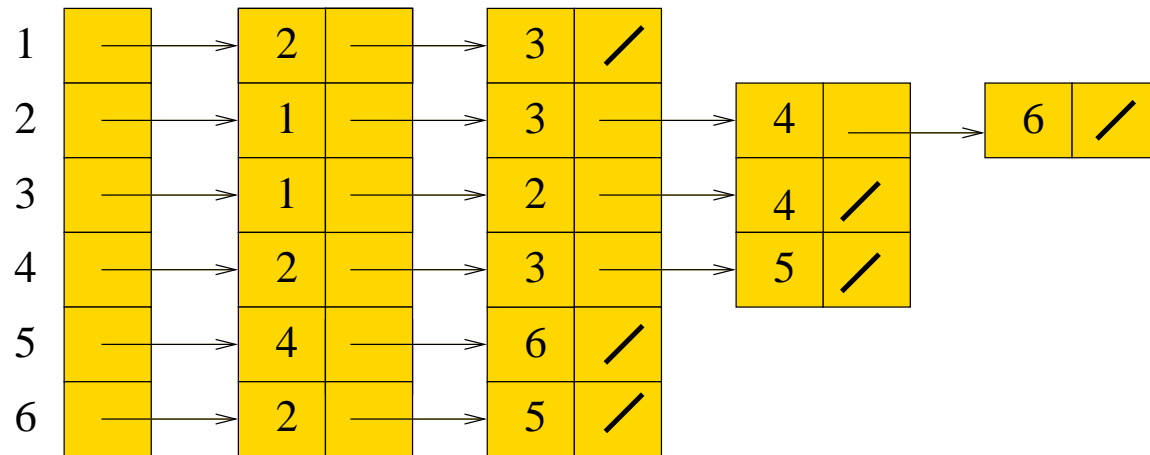
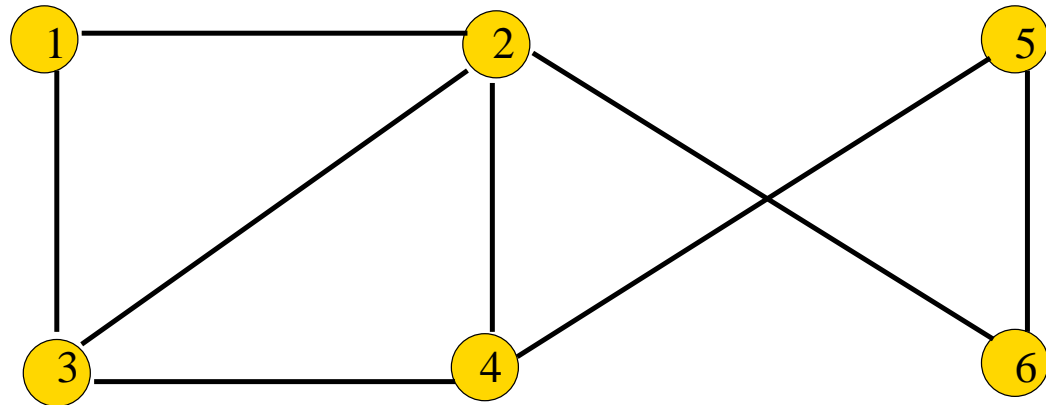
$v_1 : v_2, v_4$

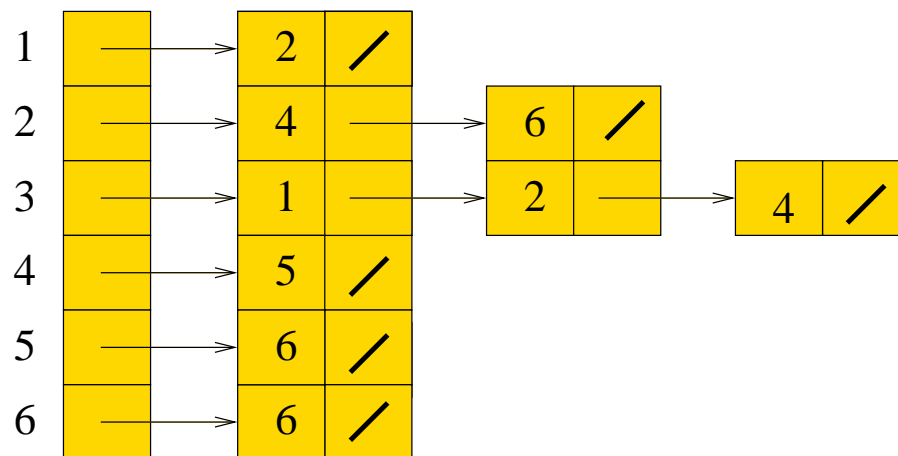
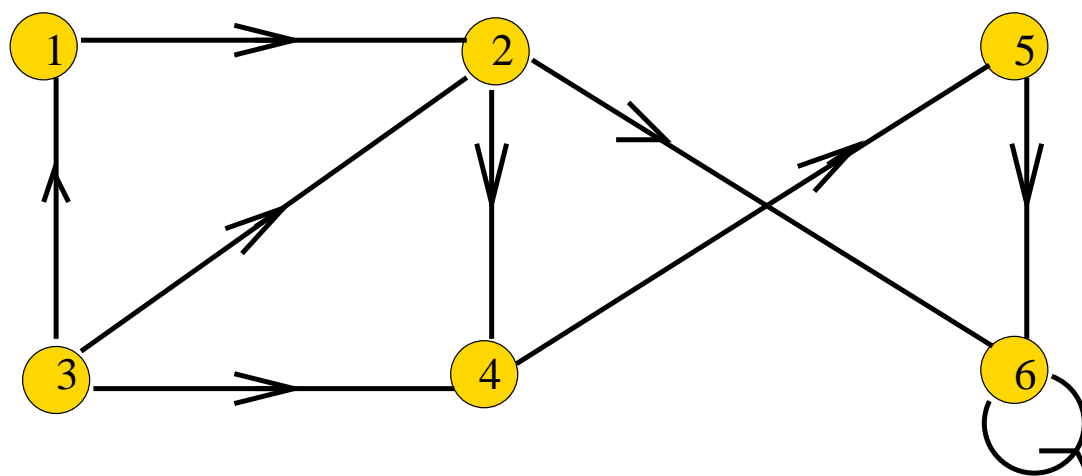
$v_2 : v_1, v_3, v_4, v_5$

$v_3 : v_2, v_4$

$v_4 : v_1, v_2, v_3, v_5$

$v_5 : v_2, v_4$





3. Przeszukiwanie grafów

- Znajdując się w pewnym wierzchołku v przeszukujemy wszystkie krawędzie incydentne do v , a następnie poruszamy się do pewnego wierzchołka przyległego w . W wierzchołku w przeszukujemy wszystkie krawędzie incydentne do w . Ten proces prowadzi się dotąd, aż przeszuka się wszystkie wierzchołki w grafie. Metodę tę nazywa się **przeszukiwaniem wszerz** (ang. *breadth-first search*, często oznaczane skrótowo **BFS**).

- zamiast przeszukiwać każdą krawędź incydentną do wierzchołka v , poruszamy się do pewnego wierzchołka przyległego w (wierzchołka, w którym dotychczas jeszcze nie byliśmy), gdy tylko to jest możliwe, pozostawiając na razie wierzchołek v z być może niezbadanymi krawędziami. Inaczej mówiąc, podążamy przez graf ścieżką przechodząc do nowego wierzchołka, gdy tylko to jest możliwe. Taka metoda przeszukiwania grafu, zwana **przeszukiwaniem w głąb** (ang. *depth-first search*, w skrócie **DFS**) lub metodą powrotu po tej samej ścieżce na grafie.
- upraszcza wiele algorytmów teorii grafów, ze względu na otrzymywane ponumerowanie wierzchołków i skierowanie krawędzi.

3.1 Przeszukiwanie grafu wszerz (BFS)

- $G = (V, E)$ - graf nieskierowanym reprezentowanym w postaci listy wierzchołków sąsiednich.
- x - ustalony wierzchołek z którego należy rozpocząć przeszukiwanie grafu
- I_v - tablica zawierająca wierzchołki incydentne z v
- NI_v liczba elementów w I_v

1. Ustaw: $Numer[x] \leftarrow 1$, $Drzewo \leftarrow \emptyset$, $Pozostale \leftarrow \emptyset$;
dopisz x do $Kolejki$.
2. Jeżeli $Kolejka$ jest pusta to STOP.
3. Pobierz element z $Kolejki$ i zapisz go jako v .
4. Dla każdego wierzchołka w incydującego do v wykonaj:
 - (a) Jeżeli $Numer[w] = 0$, tzn. wierzchołek w odwiedzamy po raz pierwszy, to nadaj wierzchołkowi w kolejny numer, dopisz w do $Kolejki$, a krawędź vw dodaj do $Drzewo$.
 - (b) Jeżeli $Numer[w] \neq 0$, tzn. wierzchołek w już był odwiedzany, natomiast krawędź $vw \notin Drzewo$ to dodaj ją do zbioru $Pozostale$.
5. Wróć do kroku 2.

BFS(G, x)

```
1   $Numer[x] \leftarrow Ponumerowano \leftarrow 1$ 
2   $NKolejka \leftarrow 1; Kolejka[NKolejka] \leftarrow x$ 
3  while  $NKolejka > 0$ 
4      do  $v \leftarrow Kolejka[1]$ 
5           $NKolejka \leftarrow NKolejka - 1$ 
6      for  $i \leftarrow 1$  to  $NKolejka$ 
7          do  $Kolejka[i] \leftarrow Kolejka[i + 1]$ 
8      for  $i \leftarrow 1$  to  $NI_v$ 
9          do  $w \leftarrow I_v[i]$ 
10         if  $Numer[w] = 0$ 
11             then  $Ponumerowano \leftarrow Ponumerowano + 1$ 
12                  $Numer[w] \leftarrow Ponumerowano$ 
13                  $Drzewo \leftarrow Drzewo \cup \{vw\}$ 
14                  $NKolejka \leftarrow NKolejka + 1$ 
15                  $Kolejka[NKolejka] \leftarrow w$ 
16         else
17              $Pozostałe \leftarrow Pozostałe \cup \{vw\}$ 
18 return  $Numer, Drzewo, Pozostałe$ 
```

Algorithm 0.0.1: BFS(G, x)

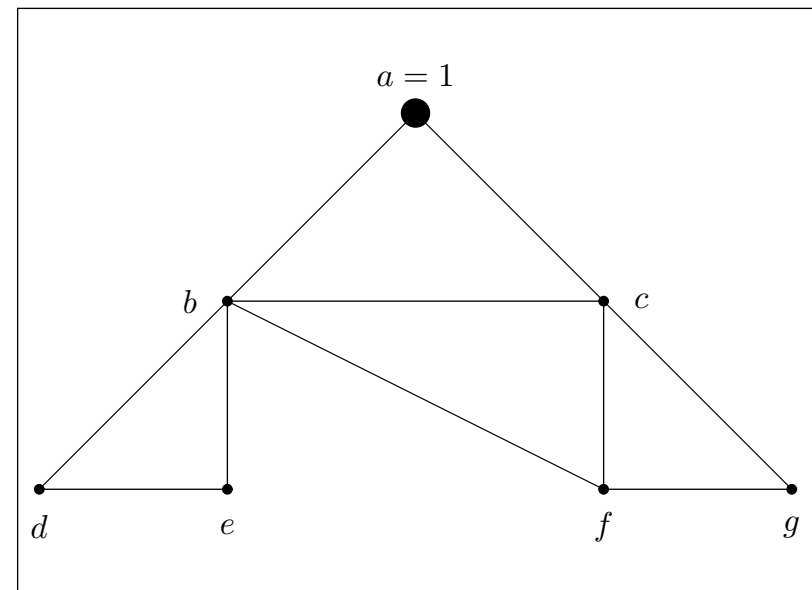
global	$Numer, Drzewo, Pozostałe$	(1)
	$Numer[x] \leftarrow Ponumerowano \leftarrow 1$	(2)
	$NKolejka \leftarrow 1; Kolejka[NKolejka] \leftarrow x$	(3)
while	$NKolejka > 0$	(4)
	$\left\{ \begin{array}{l} v \leftarrow Kolejka[1] \\ NKolejka \leftarrow NKolejka - 1 \end{array} \right.$	(5)
	$NKolejka \leftarrow NKolejka - 1$	(6)
	for $i \leftarrow 1$ to $NKolejka$	(7)
	do $Kolejka[i] \leftarrow Kolejka[i + 1]$	(8)
	for $i \leftarrow 1$ to NI_v	(9)
do	$\left\{ \begin{array}{l} w \leftarrow I_v[i] \\ \text{if } Numer[w] = 0 \end{array} \right.$	(10)
	$\text{if } Numer[w] = 0$	(11)
	$\left\{ \begin{array}{l} Ponumerowano \leftarrow Ponumerowano + 1 \\ Numer[w] \leftarrow Ponumerowano \end{array} \right.$	(12)
	$\left\{ \begin{array}{l} Numer[w] \leftarrow Ponumerowano \\ Drzewo \leftarrow Drzewo \cup \{vw\} \end{array} \right.$	(13)
	$\left\{ \begin{array}{l} Drzewo \leftarrow Drzewo \cup \{vw\} \\ NKolejka \leftarrow NKolejka + 1 \end{array} \right.$	(14)
	$\left\{ \begin{array}{l} NKolejka \leftarrow NKolejka + 1 \\ Kolejka[NKolejka] \leftarrow w \end{array} \right.$	(15)
	$\left\{ \begin{array}{l} Kolejka[NKolejka] \leftarrow w \\ \text{else } Pozostałe \leftarrow Pozostałe \cup \{vw\} \end{array} \right.$	(16)
	$\text{else } Pozostałe \leftarrow Pozostałe \cup \{vw\}$	(17)
return	$(Numer, Drzewo, Pozostałe)$	(18)

Przykład . Przeszukiwanie grafu wszerz

Algorithm 1.1: BFS(G, x)

```
global Numer, Drzewo, Pozostałe
Numer[x] ← Ponumerow ← 1
NKolejka ← 1; Kolejka[NKolejka] ← x
while NKolejka > 0
do {
  v ← Kolejka[1]
  NKolejka ← NKolejka - 1
  for i ← 1 to NKolejka
  do Kolejka[i] ← Kolejka[i + 1]
  for i ← 1 to NIv
  do {
    w ← Iv[i]
    if Numer[w] = 0
    then {
      Ponumerow ← Ponumerow + 1
      Numer[w] ← Ponumerow
      Drzewo ← Drzewo ∪ {vw}
      NKolejka ← NKolejka + 1
      Kolejka[NKolejka] ← w
    }
    else Pozostałe ← Pozostałe ∪ {vw}
  }
}
return (Numer, Drzewo, Pozostałe)
```

1 Inicjowanie



$Numer[a] \leftarrow 1$

$Ponumerow \leftarrow 1$

$Kolejka \leftarrow a$

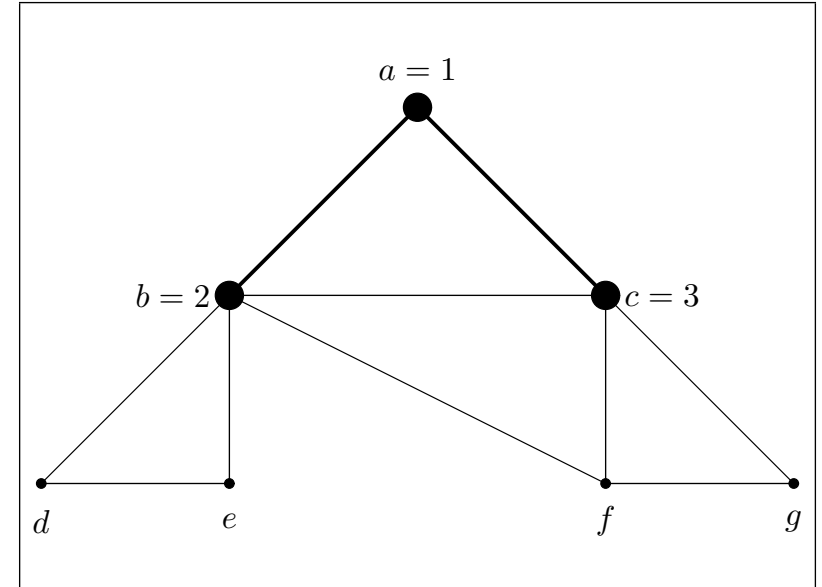
$NKolejka \leftarrow 1$

Algorithm 1.1: BFS(G, x)

```

global Numer, Drzewo, Pozostałe
Numer[ $x$ ]  $\leftarrow$  Ponumerow  $\leftarrow$  1
NKolejka  $\leftarrow$  1; Kolejka[NKolejka]  $\leftarrow$   $x$ 
while NKolejka > 0
do {
   $v \leftarrow$  Kolejka[1]
  NKolejka  $\leftarrow$  NKolejka - 1
  for  $i \leftarrow$  1 to NKolejka
  do Kolejka[ $i$ ]  $\leftarrow$  Kolejka[ $i + 1$ ]
  for  $i \leftarrow$  1 to  $NI_v$ 
  do {
     $w \leftarrow I_v[i]$ 
    if Numer[ $w$ ] = 0
    then {
      Ponumerow  $\leftarrow$  Ponumerow + 1
      Numer[ $w$ ]  $\leftarrow$  Ponumerow
      Drzewo  $\leftarrow$  Drzewo  $\cup$  { $vw$ }
      NKolejka  $\leftarrow$  NKolejka + 1
      Kolejka[NKolejka]  $\leftarrow$   $w$ 
    }
    else Pozostałe  $\leftarrow$  Pozostałe  $\cup$  { $vw$ }
  }
}
return (Numer, Drzewo, Pozostałe)

```

2 Analiza wierzchołka a 

$$v \leftarrow a \quad \text{Kolejka} \leftarrow \emptyset \quad \text{NKolejka} \leftarrow 0$$

$$w \leftarrow b$$

$$\text{Ponumerow} \leftarrow 2 \quad \text{Numer}[b] \leftarrow 2$$

$$\text{Drzewo} \leftarrow \{ab\}$$

$$\text{Kolejka} \leftarrow b \quad \text{NKolejka} \leftarrow 1$$

$$w \leftarrow c$$

$$\text{Ponumerow} \leftarrow 3 \quad \text{Numer}[c] \leftarrow 3$$

$$\text{Drzewo} \leftarrow \{ab, ac\}$$

$$\text{Kolejka} \leftarrow b, c \quad \text{NKolejka} \leftarrow 2$$

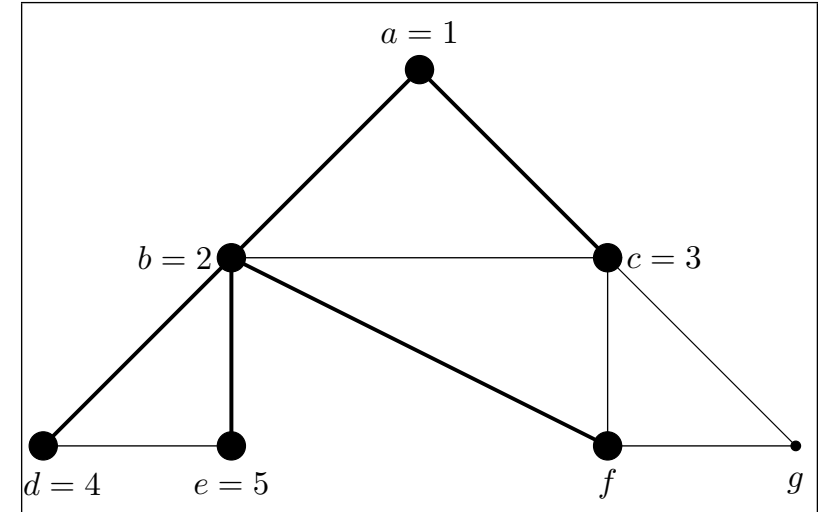
Algorithm 1.1: BFS(G, x)

```

global Numer, Drzewo, Pozostałe
Numer[x] ← Ponumerow ← 1
NKolejka ← 1; Kolejka[NKolejka] ← x
while NKolejka > 0
do {
  v ← Kolejka[1]
  NKolejka ← NKolejka - 1
  for i ← 1 to NKolejka
  do Kolejka[i] ← Kolejka[i + 1]
  for i ← 1 to NIv
  do {
    w ← Iv[i]
    if Numer[w] = 0
    then {
      Ponumerow ← Ponumerow + 1
      Numer[w] ← Ponumerow
      Drzewo ← Drzewo ∪ {vw}
      NKolejka ← NKolejka + 1
      Kolejka[NKolejka] ← w
    }
    else Pozostałe ← Pozostałe ∪ {vw}
  }
}
return (Numer, Drzewo, Pozostałe)

```

3 Analiza wierzchołka b



$v \leftarrow b \quad Kolejka \leftarrow c \quad NKolejka \leftarrow 1$

$w \leftarrow a \quad w \leftarrow c \quad w \leftarrow d$

$Ponumerow \leftarrow 4 \quad Numer[d] \leftarrow 4 \quad Drzewo \leftarrow \{ab, ac, bd, cd\}$

$Kolejka \leftarrow c, d \quad NKolejka \leftarrow 2$

$w \leftarrow e$

$Ponumerow \leftarrow 5 \quad Numer[e] \leftarrow 5 \quad Drzewo \leftarrow \{ab, ac, bd, cd, de\}$

$Kolejka \leftarrow c, d, e \quad NKolejka \leftarrow 3$

$w \leftarrow f$

$Ponumerow \leftarrow 6 \quad Numer[f] \leftarrow 6 \quad Drzewo \leftarrow \{ab, ac, bd, cd, de, ef\}$

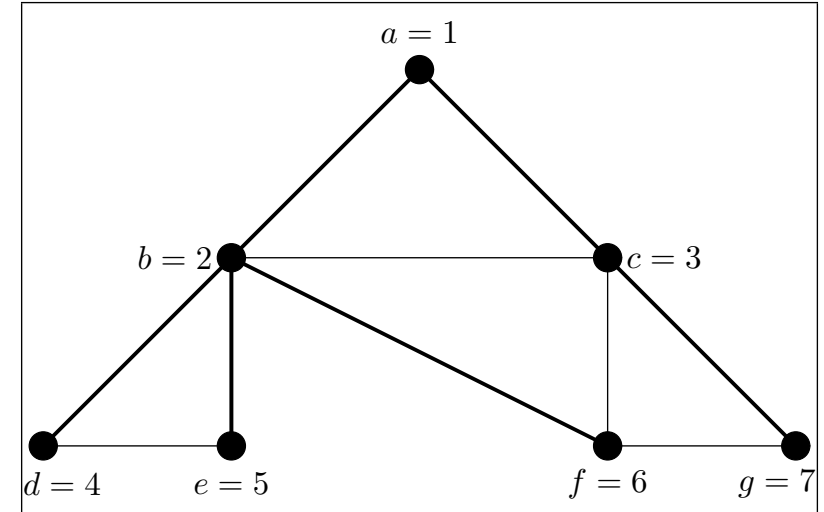
$Kolejka \leftarrow c, d, e, f \quad NKolejka \leftarrow 4$

Algorithm 1.1: BFS(G, x)

```

global Numer, Drzewo, Pozostałe
Numer[ $x$ ]  $\leftarrow$  Ponumerow  $\leftarrow$  1
NKolejka  $\leftarrow$  1; Kolejka[NKolejka]  $\leftarrow$   $x$ 
while NKolejka > 0
do {
   $v \leftarrow$  Kolejka[1]
  NKolejka  $\leftarrow$  NKolejka - 1
  for  $i \leftarrow$  1 to NKolejka
  do Kolejka[ $i$ ]  $\leftarrow$  Kolejka[ $i + 1$ ]
  for  $i \leftarrow$  1 to  $NI_v$ 
  do {
     $w \leftarrow I_v[i]$ 
    if Numer[ $w$ ] = 0
    then {
      Ponumerow  $\leftarrow$  Ponumerow + 1
      Numer[ $w$ ]  $\leftarrow$  Ponumerow
      Drzewo  $\leftarrow$  Drzewo  $\cup$  { $vw$ }
      NKolejka  $\leftarrow$  NKolejka + 1
      Kolejka[NKolejka]  $\leftarrow$   $w$ 
    }
    else Pozostałe  $\leftarrow$  Pozostałe  $\cup$  { $vw$ }
  }
}
return (Numer, Drzewo, Pozostałe)

```

4 Analiza wierzchołka c 

$$v \leftarrow c \quad \text{Kolejka} \leftarrow d, e, f \quad \text{NKolejka} \leftarrow 3$$

$$w \leftarrow a \quad w \leftarrow b \quad w \leftarrow f \quad w \leftarrow g$$

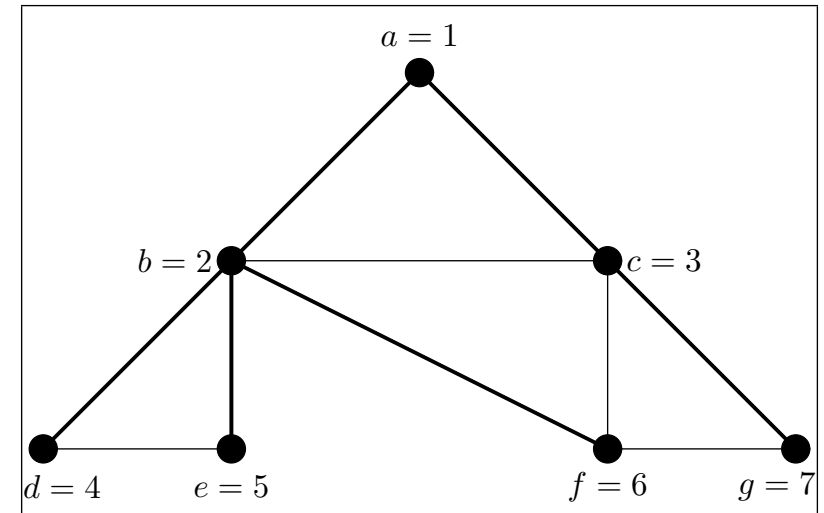
$$\text{Ponumerow} \leftarrow 7 \quad \text{Numer}[g] \leftarrow 7$$

$$\text{Drzewo} \leftarrow \{ab, ac, bd, be, bf, cg\}$$

$$\text{Kolejka} \leftarrow d, e, f, g \quad \text{NKolejka} \leftarrow 4$$

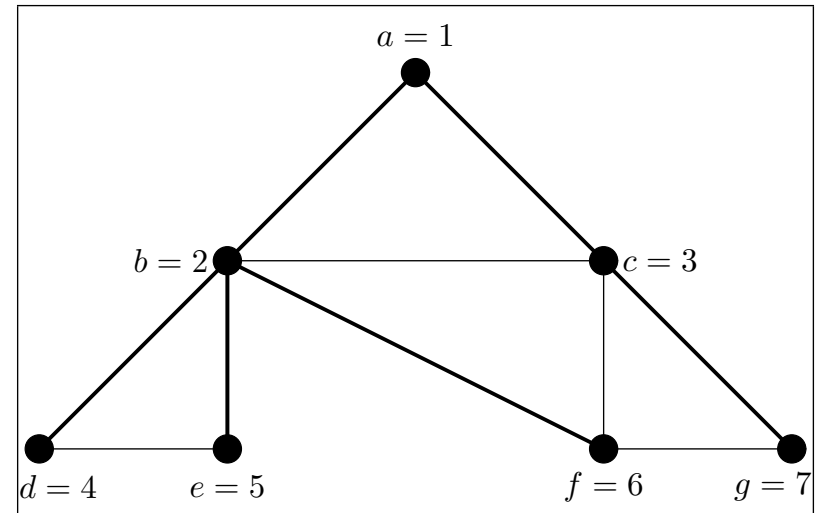
Algorithm 1.1: BFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NKolejka \leftarrow 1; Kolejka[NKolejka] \leftarrow x$ **while** $NKolejka > 0$

do {	{	$v \leftarrow Kolejka[1]$	
		$NKolejka \leftarrow NKolejka - 1$	
		for $i \leftarrow 1$ to $NKolejka$	
		do $Kolejka[i] \leftarrow Kolejka[i + 1]$	
		for $i \leftarrow 1$ to NI_v	
		{	$w \leftarrow I_v[i]$
	if $Numer[w] = 0$		
	$Ponumerow \leftarrow Ponumerow + 1$		
	$Numer[w] \leftarrow Ponumerow$		
	$Drzewo \leftarrow Drzewo \cup \{vw\}$		
			$NKolejka \leftarrow NKolejka + 1$
			$Kolejka[NKolejka] \leftarrow w$
			else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$
			end

return ($Numer, Drzewo, Pozostałe$)**5 Analiza wierzchołka d**  $v \leftarrow d \quad Kolejka \leftarrow e, f, g \quad NKolejka \leftarrow 3$ $w \leftarrow b \quad w \leftarrow e$

Algorithm 1.1: BFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NKolejka \leftarrow 1; Kolejka[NKolejka] \leftarrow x$ **while** $NKolejka > 0$

do {	{	$v \leftarrow Kolejka[1]$	
		$NKolejka \leftarrow NKolejka - 1$	
		for $i \leftarrow 1$ to $NKolejka$	
		do $Kolejka[i] \leftarrow Kolejka[i + 1]$	
		for $i \leftarrow 1$ to NI_v	
		{	$w \leftarrow I_v[i]$
	if $Numer[w] = 0$		
	$Ponumerow \leftarrow Ponumerow + 1$		
	$Numer[w] \leftarrow Ponumerow$		
	$Drzewo \leftarrow Drzewo \cup \{vw\}$		
			$NKolejka \leftarrow NKolejka + 1$
			$Kolejka[NKolejka] \leftarrow w$
			else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$
			return ($Numer, Drzewo, Pozostałe$)

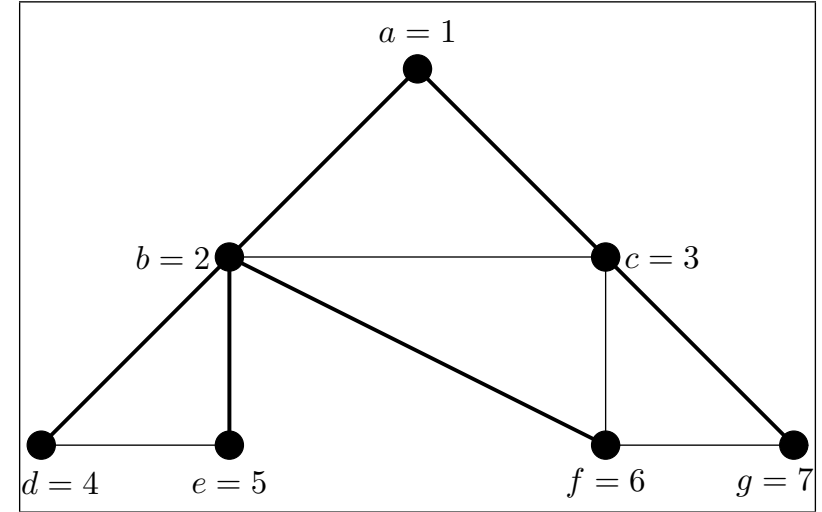
6 Analiza wierzchołka e  $v \leftarrow e \quad Kolejka \leftarrow f, g \quad NKolejka \leftarrow 2$ $w \leftarrow b \quad w \leftarrow d \quad w \leftarrow f$

Algorithm 1.1: BFS(G, x)

```

global Numer, Drzewo, Pozostałe
Numer[ $x$ ]  $\leftarrow$  Ponumerow  $\leftarrow$  1
NKolejka  $\leftarrow$  1; Kolejka[NKolejka]  $\leftarrow$   $x$ 
while NKolejka > 0
do {
   $v \leftarrow$  Kolejka[1]
  NKolejka  $\leftarrow$  NKolejka - 1
  for  $i \leftarrow$  1 to NKolejka
  do Kolejka[ $i$ ]  $\leftarrow$  Kolejka[ $i + 1$ ]
  for  $i \leftarrow$  1 to  $NI_v$ 
  do {
     $w \leftarrow I_v[i]$ 
    if Numer[ $w$ ] = 0
    then {
      Ponumerow  $\leftarrow$  Ponumerow + 1
      Numer[ $w$ ]  $\leftarrow$  Ponumerow
      Drzewo  $\leftarrow$  Drzewo  $\cup$  { $vw$ }
      NKolejka  $\leftarrow$  NKolejka + 1
      Kolejka[NKolejka]  $\leftarrow$   $w$ 
    }
    else Pozostałe  $\leftarrow$  Pozostałe  $\cup$  { $vw$ }
  }
}
return (Numer, Drzewo, Pozostałe)

```

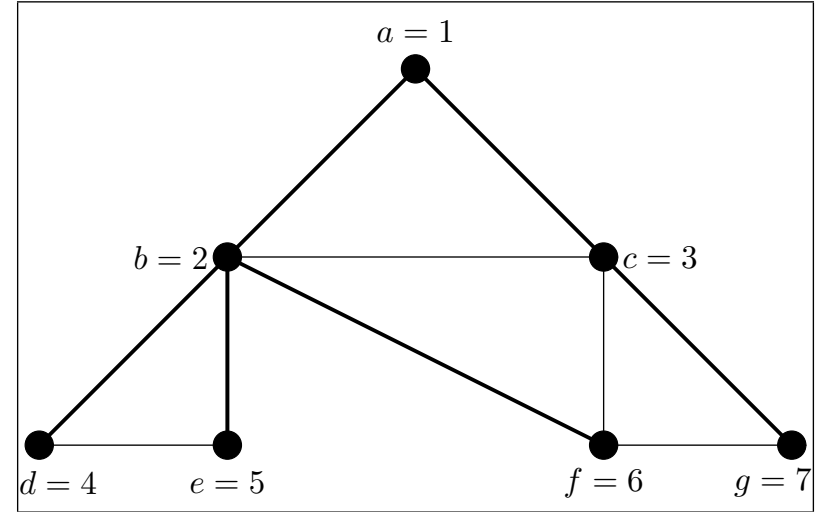
7 Analiza wierzchołka f 

$$v \leftarrow f \quad \text{Kolejka} \leftarrow g \quad \text{NKolejka} \leftarrow 1$$

$$w \leftarrow b \quad w \leftarrow c \quad w \leftarrow g$$

Algorithm 1.1: BFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NKolejka \leftarrow 1; Kolejka[NKolejka] \leftarrow x$ **while** $NKolejka > 0$

do {	{	$v \leftarrow Kolejka[1]$	
		$NKolejka \leftarrow NKolejka - 1$	
		for $i \leftarrow 1$ to $NKolejka$	
		do $Kolejka[i] \leftarrow Kolejka[i + 1]$	
		for $i \leftarrow 1$ to NI_v	
	do {	{	$w \leftarrow I_v[i]$
if $Numer[w] = 0$			
then {			
$Ponumerow \leftarrow Ponumerow + 1$			
$Numer[w] \leftarrow Ponumerow$			
		then {	$Drzewo \leftarrow Drzewo \cup \{vw\}$
	$NKolejka \leftarrow NKolejka + 1$		
	$Kolejka[NKolejka] \leftarrow w$		
		else	$Pozostałe \leftarrow Pozostałe \cup \{vw\}$

return ($Numer, Drzewo, Pozostałe$)**8 Analiza wierzchołka g**  $v \leftarrow g \quad Kolejka \leftarrow \emptyset \quad NKolejka \leftarrow 0$ $w \leftarrow c \quad w \leftarrow f$

3.2 Przeszukiwanie grafu w głąb (DFS)

- $G = (V, E)$ - graf nieskierowanym reprezentowanym w postaci listy wierzchołków sąsiednich.
- x - ustalony wierzchołek z którego należy rozpocząć przeszukiwanie grafu
- I_v - tablica zawierająca wierzchołki incydentne z v , NI_v liczba elementów w I_v
- $Stos$ - tablica przechowująca ciąg wierzchołków umożliwiającą "powrót", $NStos$ - liczba wierzchołków w $Stos$

1. Ustaw $v \leftarrow x$, $i \leftarrow 0$, $Drzewo \leftarrow \emptyset$, $Pozostale \leftarrow \emptyset$.

2. Ustaw $i \leftarrow i + 1$, $Numer(v) \leftarrow i$.

3. Poszukaj nieprzebytej krawędzi incydentnej do wierzchołka v .

- Jeżeli nie ma takiej krawędzi (tzn. po każdej krawędzi incydentnej do v już przeszliśmy), to przejdź do kroku 5.
- Wybierz pierwszą nieprzebytą krawędź incydentną do wierzchołka v , powiedzmy vw i przejdź ją.

4. Jesteśmy teraz w wierzchołku w .

- Jeżeli w jest wierzchołkiem, w którym jeszcze nie byliśmy podczas tego szukania (tzn. $Numer(w)$ jest nieokreślony), to dodaj krawędź vw do zbioru *Drzewo*. Ustaw $v \leftarrow w$ i przejdź do kroku 2.
- Jeżeli w jest wierzchołkiem, w którym już wcześniej byliśmy (tzn. $Numer(w) < Numer(v)$), to dodaj krawędź vw do zbioru *Pozostale*. Przejdź do kroku 3. Jesteśmy więc z powrotem w wierzchołku v .

5. Sprawdź, czy istnieje jakaś przebyta krawędź uv w zbiorze *Drzewo* z $Numer(u) < Numer(v)$.

- Jeżeli jest taka krawędź, to wróć do wierzchołka u . (Zauważmy, że u jest wierzchołkiem, z którego osiągnięto v po raz pierwszy). Ustaw $v \leftarrow u$ i przejdź do kroku 3.
- Jeżeli nie ma takiej krawędzi, to zatrzymaj algorytm (jesteśmy z powrotem w korzeniu x po przejściu każdej krawędzi i odwiedzeniu każdego wierzchołka połączonego z x).

DFS(G, x)

```
1   $Numer[x] \leftarrow Ponumerow \leftarrow 1$ 
2   $NStos \leftarrow 1$    $Stos[NStos] \leftarrow x$ 
3  while  $NStos > 0$ 
4      do  $v \leftarrow Stos[NStos]$ 
5          if  $NI_v = 0$ 
6              then  $NStos \leftarrow NStos - 1$ 
7              else  $w \leftarrow I_v[1]$ 
8                   $NI_v \leftarrow NI_v - 1$ 
9                  for  $i \leftarrow 1$  to  $NI_v$ 
10                      do  $I_v[i] \leftarrow I_v[i - 1]$ 
11                  if  $Numer[w] = 0$ 
12                      then  $Ponumerow \leftarrow Ponumerow + 1$ 
13                       $Numer[w] \leftarrow Ponumerow$ 
14                       $Drzewo \leftarrow Drzewo \cup \{vw\}$ 
15                       $NStos \leftarrow NStos + 1$ 
16                       $Stos[NStos] \leftarrow w$ 
17                  else  $Pozostale \leftarrow Pozostale \cup \{vw\}$ 
18 return  $Numer, Drzewo, Pozostale$ 
```

Algorithm 0.0.1: DFS(G, x)

```
global  $Numer, Drzewo, Pozostałe$  (1)
 $Numer[x] \leftarrow Ponumerow \leftarrow 1$  (2)
 $NStos \leftarrow 1; Stos[NStos] \leftarrow x$  (3)
while  $NStos > 0$  (4)
    do { (5)
         $v \leftarrow STos[NStos]$  (6)
        if  $NI_v = 0$  (7)
            then  $NStos \leftarrow NStos - 1$  (8)
            { (9)
                 $w \leftarrow I_v[1]$  (10)
                 $NI_v \leftarrow NI_v - 1$  (11)
                for  $i \leftarrow 1$  to  $NI_v$  (12)
                    do  $I_v[i] \leftarrow I_v[i - 1]$  (13)
                if  $Numer[w] = 0$  (14)
                    { (15)
                         $Ponumerow \leftarrow Ponumerow + 1$  (16)
                         $Numer[w] \leftarrow Ponumerow$  (17)
                        then { (18)
                             $Drzewo \leftarrow Drzewo \cup \{vw\}$  (19)
                             $NStos \leftarrow NStos + 1$  (20)
                             $Stos[NStos] \leftarrow w$  (21)
                        }
                    }
                else  $Pozostałe \leftarrow Pozostałe \cup \{vw\}$  (22)
            }
        }
return ( $Numer, Drzewo, Pozostałe$ ) (23)
```

Przykład . Przeszukiwanie grafu w głąb

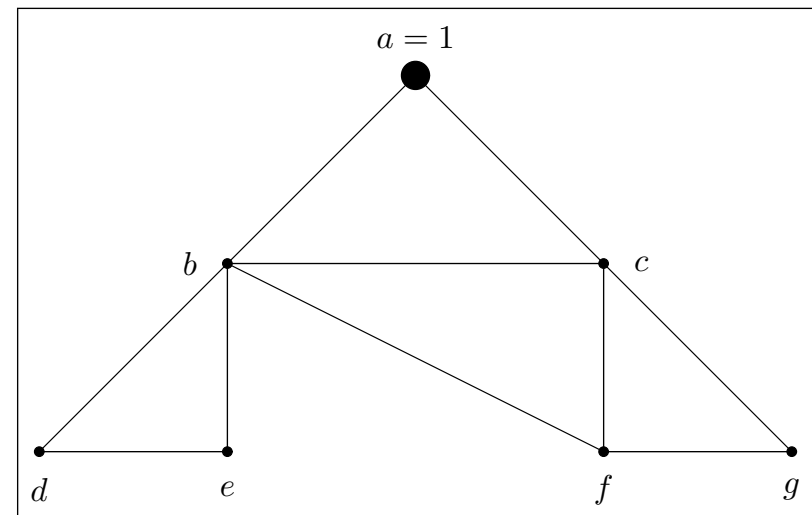
Algorithm 1.1: DFS(G, x)

```

global Numer, Drzewo, Pozostałe
Numer[x] ← Ponumerow ← 1
NStos ← 1; Stos[NStos] ← x
while NStos > 0
do {
  v ← STos[NStos]
  if NIv = 0
  then NStos ← NStos - 1
  else {
    w ← Iv[1]
    NIv ← NIv - 1
    for i ← 1 to NIv
    do Iv[i] ← Iv[i - 1]
    if Numer[w] = 0
    then {
      Ponumerow ← Ponumerow + 1
      Numer[w] ← Ponumerow
      Drzewo ← Drzewo ∪ {vw}
      NStos ← NStos + 1
      Stos[NStos] ← w
    }
    else Pozostałe ← Pozostałe ∪ {vw}
  }
}
return (Numer, Drzewo, Pozostałe)

```

1 Inicjowanie



$Numer[a] \leftarrow 1$

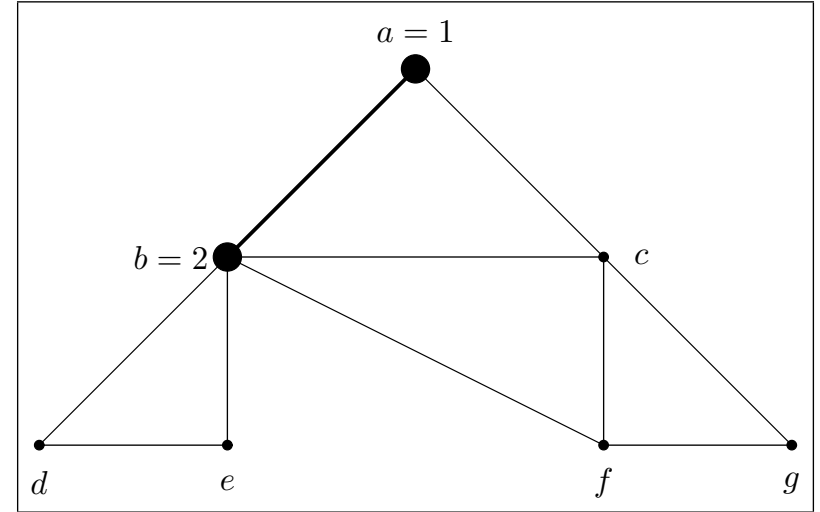
$Ponumerow \leftarrow 1$

$Stos \leftarrow [a]$

$NStos \leftarrow 1$

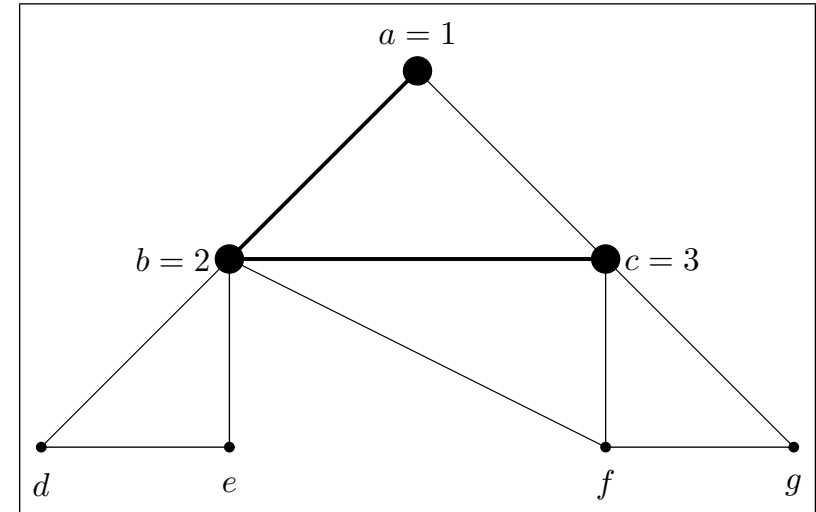
Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$
		$NI_v \leftarrow NI_v - 1$
		for $i \leftarrow 1$ to NI_v
		do $I_v[i] \leftarrow I_v[i - 1]$
		if $Numer[w] = 0$
	{	$Ponumerow \leftarrow Ponumerow + 1$
		$Numer[w] \leftarrow Ponumerow$
		then {
		$Drzewo \leftarrow Drzewo \cup \{vw\}$
		$NStos \leftarrow NStos + 1$
		$Stos[NStos] \leftarrow w$
		else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$

return ($Numer, Drzewo, Pozostałe$)**2 Wyjście z a** $v \leftarrow a$ $w \leftarrow b$ $Ponumerow \leftarrow 2$ $Numer[b] \leftarrow 2$ $Drzewo \leftarrow \{ab\}$ $NStos \leftarrow 2$ $Stos \leftarrow [a, b]$

Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$ $NI_v \leftarrow NI_v - 1$ for $i \leftarrow 1$ to NI_v do $I_v[i] \leftarrow I_v[i - 1]$ if $Numer[w] = 0$

return ($Numer, Drzewo, Pozostałe$)**3 Wyjście z wierzchołka b**  $v \leftarrow b$ $w \leftarrow a$ $w \leftarrow c$ $Ponumerow \leftarrow 3$ $Numer[c] \leftarrow 3$ $Drzewo \leftarrow \{ab, bc\}$ $NStos \leftarrow 3$ $Stos \leftarrow [a, b, c]$

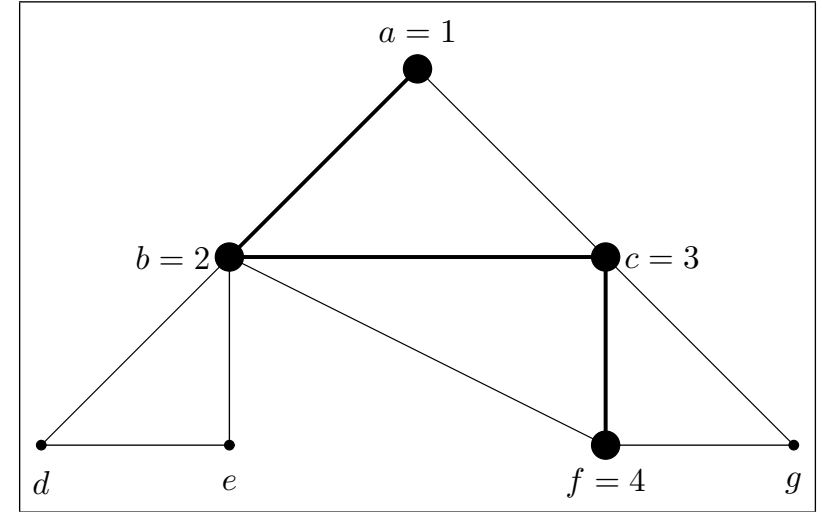
Algorithm 1.1: DFS(G, x)

```

global Numer, Drzewo, Pozostałe
Numer[x] ← Ponumerow ← 1
NStos ← 1; Stos[NStos] ← x
while NStos > 0
do {
  v ← STos[NStos]
  if NIv = 0
  then NStos ← NStos - 1
  else {
    w ← Iv[1]
    NIv ← NIv - 1
    for i ← 1 to NIv
    do Iv[i] ← Iv[i - 1]
    if Numer[w] = 0
    then {
      Ponumerow ← Ponumerow + 1
      Numer[w] ← Ponumerow
      Drzewo ← Drzewo ∪ {vw}
      NStos ← NStos + 1
      Stos[NStos] ← w
    }
    else Pozostałe ← Pozostałe ∪ {vw}
  }
}
return (Numer, Drzewo, Pozostałe)

```

4 Wyjście z wierzchołka c



$v \leftarrow c$

$w \leftarrow a \quad w \leftarrow b \quad w \leftarrow f$

$Ponumerow \leftarrow 4$

$Numer[f] \leftarrow 3$

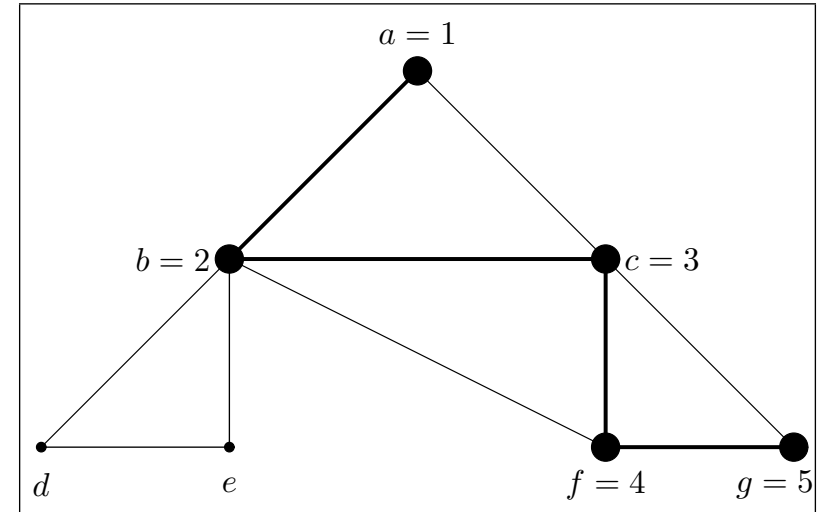
$Drzewo \leftarrow \{ab, bc, cf\}$

$NStos \leftarrow 4$

$Stos \leftarrow [a, b, c, f]$

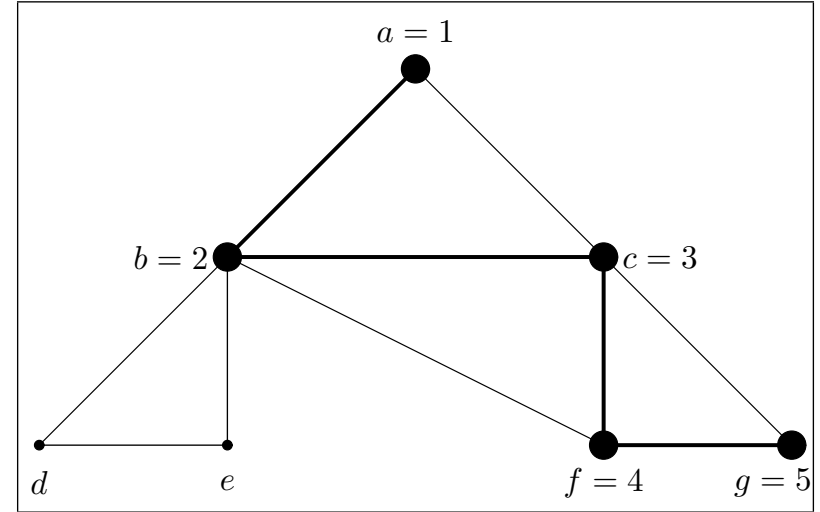
Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$
		$NI_v \leftarrow NI_v - 1$
		for $i \leftarrow 1$ to NI_v
		do $I_v[i] \leftarrow I_v[i - 1]$
		if $Numer[w] = 0$
	{	$Ponumerow \leftarrow Ponumerow + 1$
		$Numer[w] \leftarrow Ponumerow$
		then $Drzewo \leftarrow Drzewo \cup \{vw\}$
		$NStos \leftarrow NStos + 1$
		$Stos[NStos] \leftarrow w$
		else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$
return ($Numer, Drzewo, Pozostałe$)		

5 Wyjście z wierzchołka f  $v \leftarrow f$ $w \leftarrow b$ $w \leftarrow c$ $w \leftarrow g$ $Ponumerow \leftarrow 5$ $Numer[g] \leftarrow 5$ $Drzewo \leftarrow \{ab, bc, cf, fg\}$ $NStos \leftarrow 5$ $Stos \leftarrow [a, b, c, f, g]$

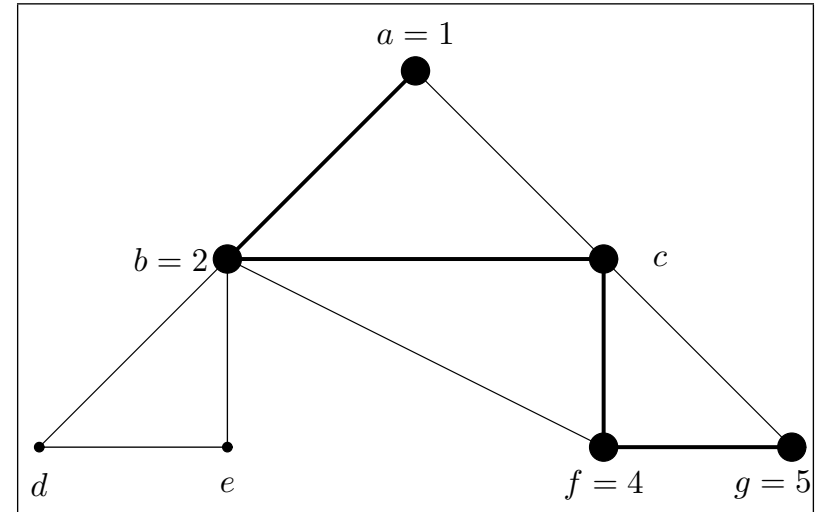
Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

{	do	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$
		$NI_v \leftarrow NI_v - 1$
		for $i \leftarrow 1$ to NI_v
		do $I_v[i] \leftarrow I_v[i - 1]$
		if $Numer[w] = 0$
	{	$Ponumerow \leftarrow Ponumerow + 1$
		$Numer[w] \leftarrow Ponumerow$
		then $Drzewo \leftarrow Drzewo \cup \{vw\}$
		$NStos \leftarrow NStos + 1$
		$Stos[NStos] \leftarrow w$
		else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$

return ($Numer, Drzewo, Pozostałe$)**6 Wycofanie się z wierzchołka g**  $v \leftarrow g$ $w \leftarrow c$ $w \leftarrow f$ $NStos \leftarrow 4$ $Stos \leftarrow [a, b, c, f]$

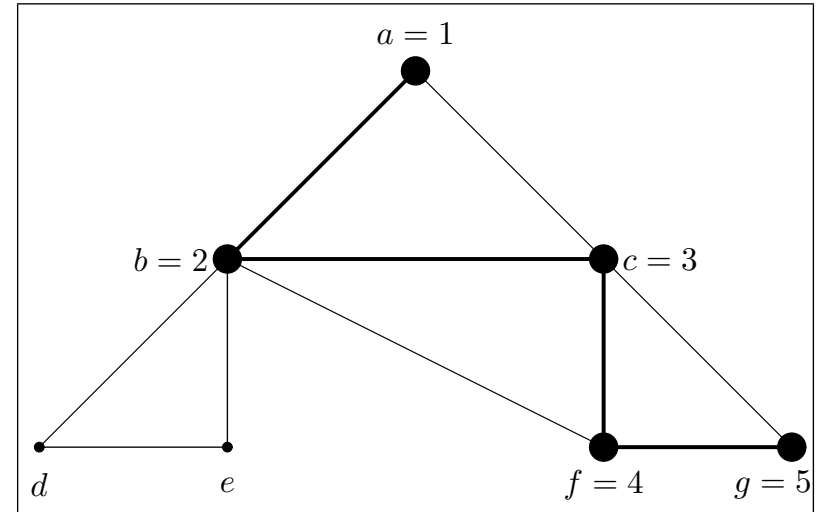
Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$
		$NI_v \leftarrow NI_v - 1$
		for $i \leftarrow 1$ to NI_v
		do $I_v[i] \leftarrow I_v[i - 1]$
		if $Numer[w] = 0$
	{	$Ponumerow \leftarrow Ponumerow + 1$
		$Numer[w] \leftarrow Ponumerow$
		then {
		$Drzewo \leftarrow Drzewo \cup \{vw\}$
		$NStos \leftarrow NStos + 1$
		$Stos[NStos] \leftarrow w$
		else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$
return ($Numer, Drzewo, Pozostałe$)		

7 Wycofanie się z wierzchołka f  $v \leftarrow f$ $NStos \leftarrow 3$ $Stos \leftarrow [a, b, c]$

Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$
		$NI_v \leftarrow NI_v - 1$
		for $i \leftarrow 1$ to NI_v
		do $I_v[i] \leftarrow I_v[i - 1]$
		if $Numer[w] = 0$
	{	$Ponumerow \leftarrow Ponumerow + 1$
		$Numer[w] \leftarrow Ponumerow$
then {		$Drzewo \leftarrow Drzewo \cup \{vw\}$
$NStos \leftarrow NStos + 1$		
		$Stos[NStos] \leftarrow w$
		else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$
return ($Numer, Drzewo, Pozostałe$)		

8 Wycofanie się z wierzchołka c  $v \leftarrow c$ $w \leftarrow g$ $NStos \leftarrow 2$ $Stos \leftarrow [a, b]$

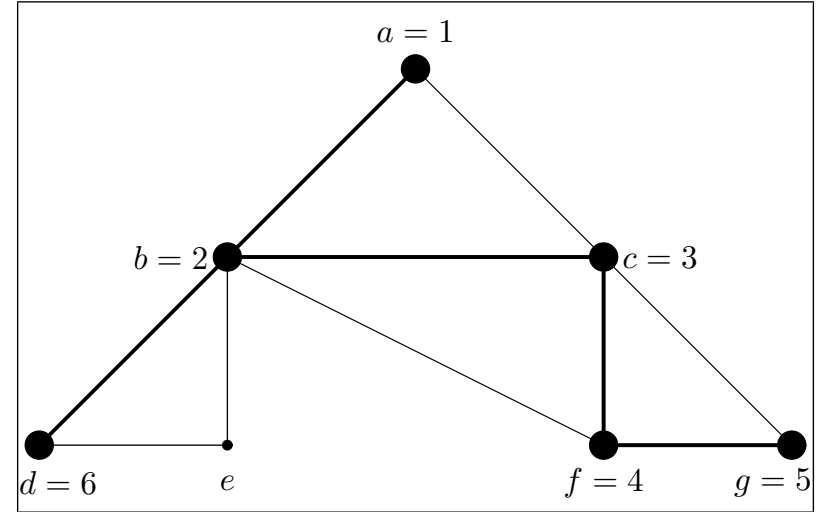
Algorithm 1.1: DFS(G, x)

```

global Numer, Drzewo, Pozostałe
Numer[x] ← Ponumerow ← 1
NStos ← 1; Stos[NStos] ← x
while NStos > 0
do {
  v ← STos[NStos]
  if NIv = 0
  then NStos ← NStos - 1
  else {
    w ← Iv[1]
    NIv ← NIv - 1
    for i ← 1 to NIv
    do Iv[i] ← Iv[i - 1]
    if Numer[w] = 0
    then {
      Ponumerow ← Ponumerow + 1
      Numer[w] ← Ponumerow
      Drzewo ← Drzewo ∪ {vw}
      NStos ← NStos + 1
      Stos[NStos] ← w
    }
    else Pozostałe ← Pozostałe ∪ {vw}
  }
}
return (Numer, Drzewo, Pozostałe)

```

9 Wyjście z wierzchołka b



$v \leftarrow b$

$w \leftarrow d$

$Ponumerow \leftarrow 6$

$Numer[d] \leftarrow 5$

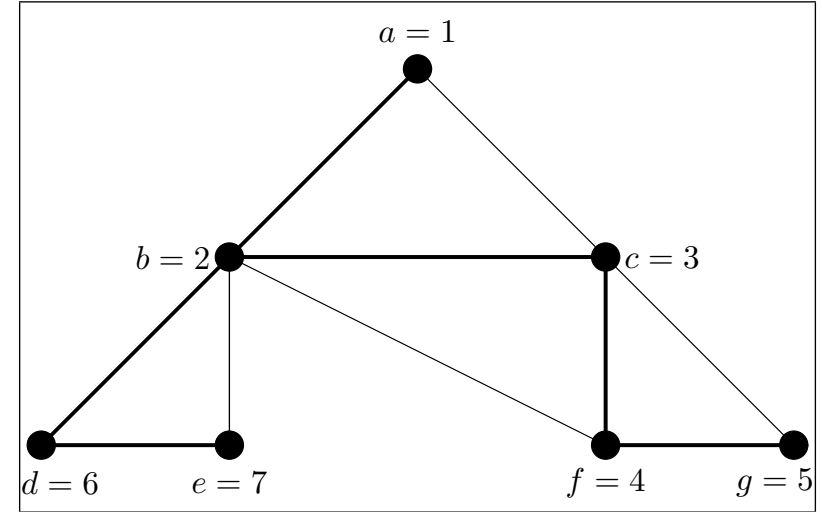
$Drzewo \leftarrow \{ab, bc, cf, fg, bd\}$

$NStos \leftarrow 3$

$Stos \leftarrow [a, b, d]$

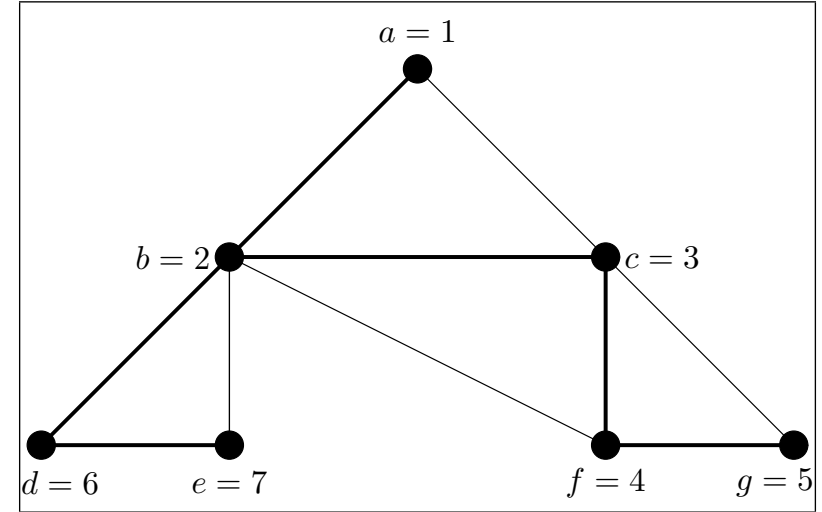
Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow STos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$
		$NI_v \leftarrow NI_v - 1$
		for $i \leftarrow 1$ to NI_v
		do $I_v[i] \leftarrow I_v[i - 1]$
		if $Numer[w] = 0$
	{	$Ponumerow \leftarrow Ponumerow + 1$
		$Numer[w] \leftarrow Ponumerow$
		then {
		$Drzewo \leftarrow Drzewo \cup \{vw\}$
		$NStos \leftarrow NStos + 1$
		$Stos[NStos] \leftarrow w$
		else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$

return ($Numer, Drzewo, Pozostałe$)**10 Wyjście z wierzchołka d**  $v \leftarrow d$ $w \leftarrow b$ $w \leftarrow e$ $Ponumerow \leftarrow 7$ $Numer[e] \leftarrow 7$ $Drzewo \leftarrow \{ab, bc, cf, fg, bd, de\}$ $NStos \leftarrow 4$ $Stos \leftarrow [a, b, d, e]$

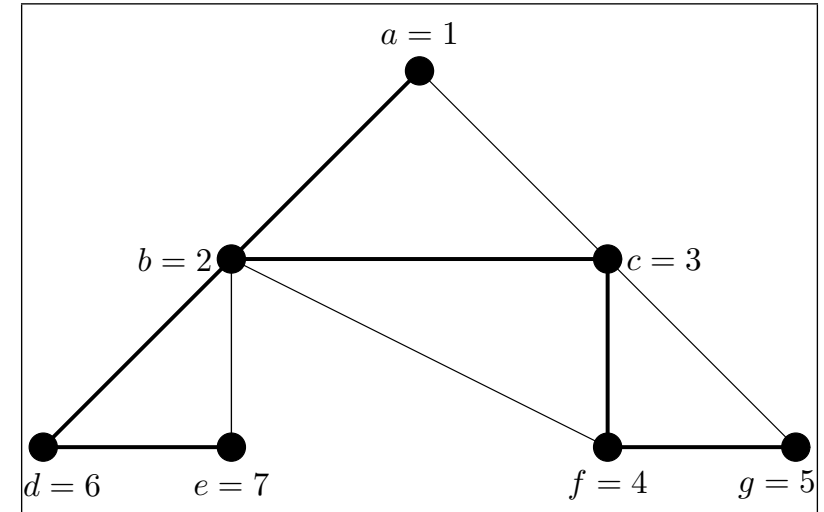
Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$
		$NI_v \leftarrow NI_v - 1$
		for $i \leftarrow 1$ to NI_v
		do $I_v[i] \leftarrow I_v[i - 1]$
		if $Numer[w] = 0$
	{	$Ponumerow \leftarrow Ponumerow + 1$
		$Numer[w] \leftarrow Ponumerow$
		then {
		$Drzewo \leftarrow Drzewo \cup \{vw\}$
		$NStos \leftarrow NStos + 1$
		$Stos[NStos] \leftarrow w$
		else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$
		return ($Numer, Drzewo, Pozostałe$)

11 Próba wyjścia z wierzchołka e  $v \leftarrow e$ $w \leftarrow b$ $w \leftarrow d$ $NStos \leftarrow 3$ $Stos \leftarrow [a, b, d]$

Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow STos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$ $NI_v \leftarrow NI_v - 1$ for $i \leftarrow 1$ to NI_v do $I_v[i] \leftarrow I_v[i - 1]$ if $Numer[w] = 0$

return ($Numer, Drzewo, Pozostałe$)**12** Próba wyjścia z wierzchołka d  $v \leftarrow d$ $NStos \leftarrow 2$ $Stos \leftarrow [a, b]$

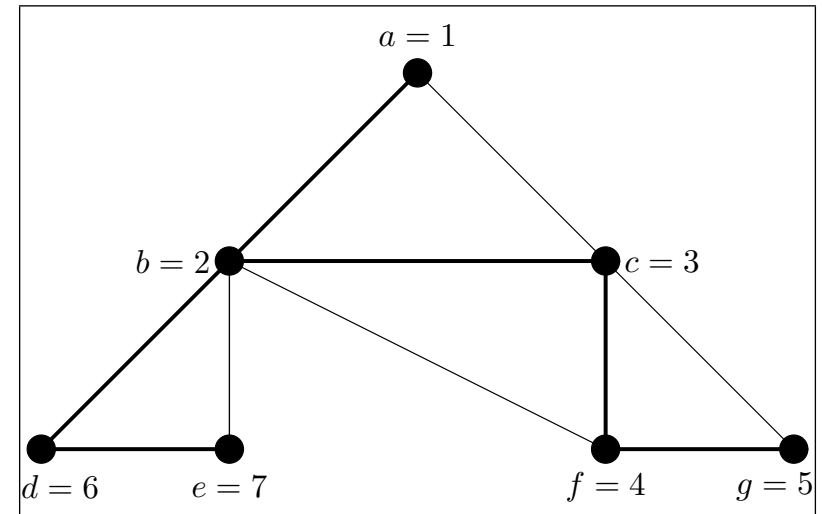
Algorithm 1.1: DFS(G, x)

```

global Numer, Drzewo, Pozostałe
Numer[x] ← Ponumerow ← 1
NStos ← 1; Stos[NStos] ← x
while NStos > 0
do {
  v ← STos[NStos]
  if NIv = 0
  then NStos ← NStos − 1
    {
      w ← Iv[1]
      NIv ← NIv − 1
      for i ← 1 to NIv
      do Iv[i] ← Iv[i − 1]
      if Numer[w] = 0
      then {
        Ponumerow ← Ponumerow + 1
        Numer[w] ← Ponumerow
        Drzewo ← Drzewo ∪ {vw}
        NStos ← NStos + 1
        Stos[NStos] ← w
      }
      else Pozostałe ← Pozostałe ∪ {vw}
    }
  return (Numer, Drzewo, Pozostałe)

```

13 Próba wyjścia z wierzchołka b



$v \leftarrow b$

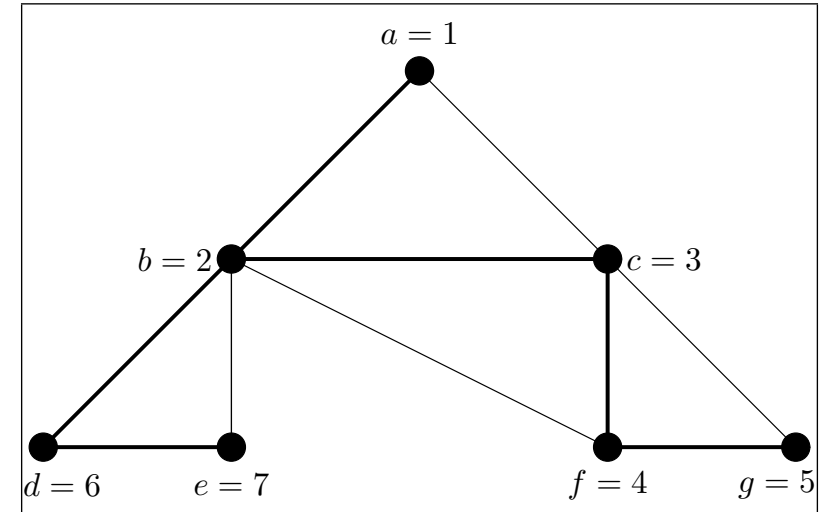
$w \leftarrow f$

$NStos \leftarrow 1$

$Stos \leftarrow [a]$

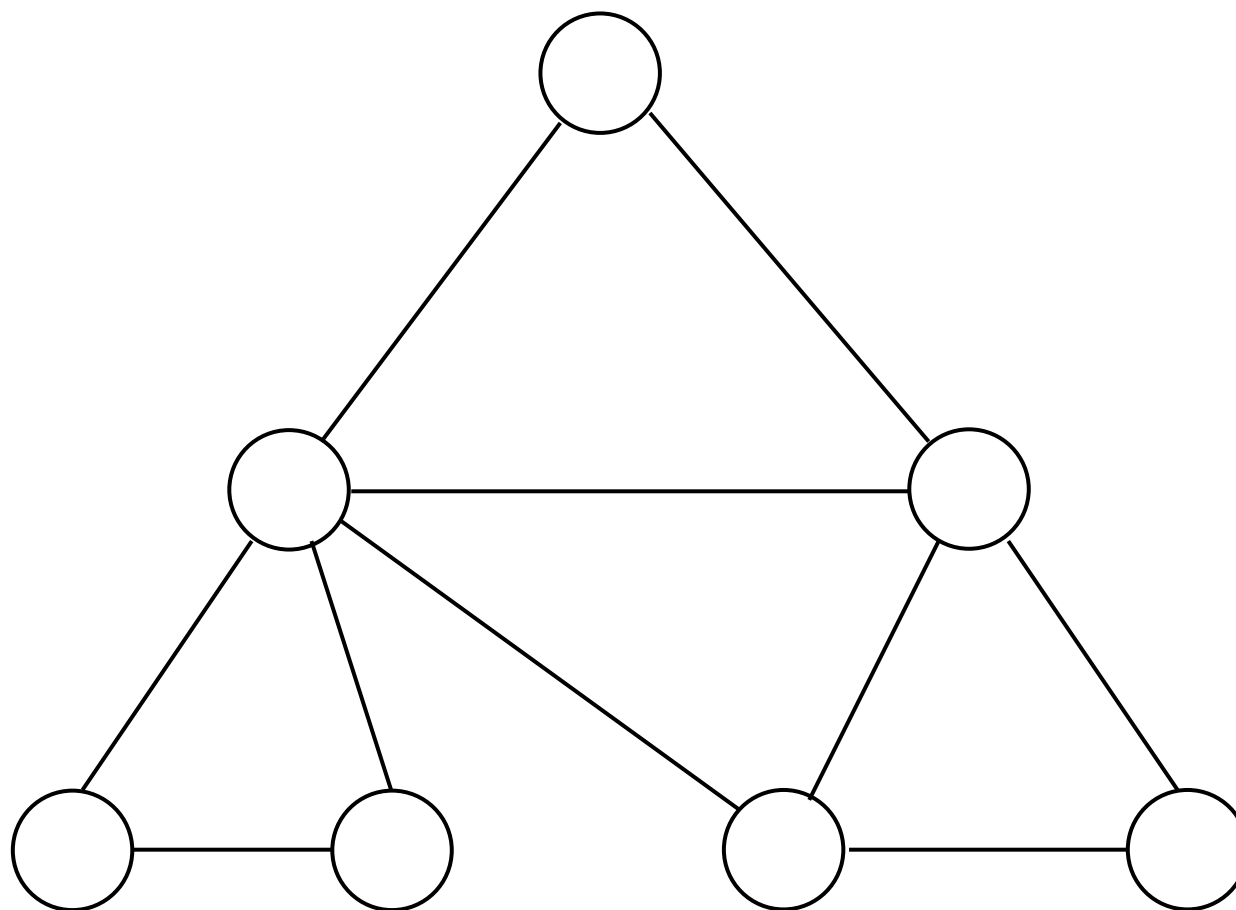
Algorithm 1.1: DFS(G, x)**global** $Numer, Drzewo, Pozostałe$ $Numer[x] \leftarrow Ponumerow \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

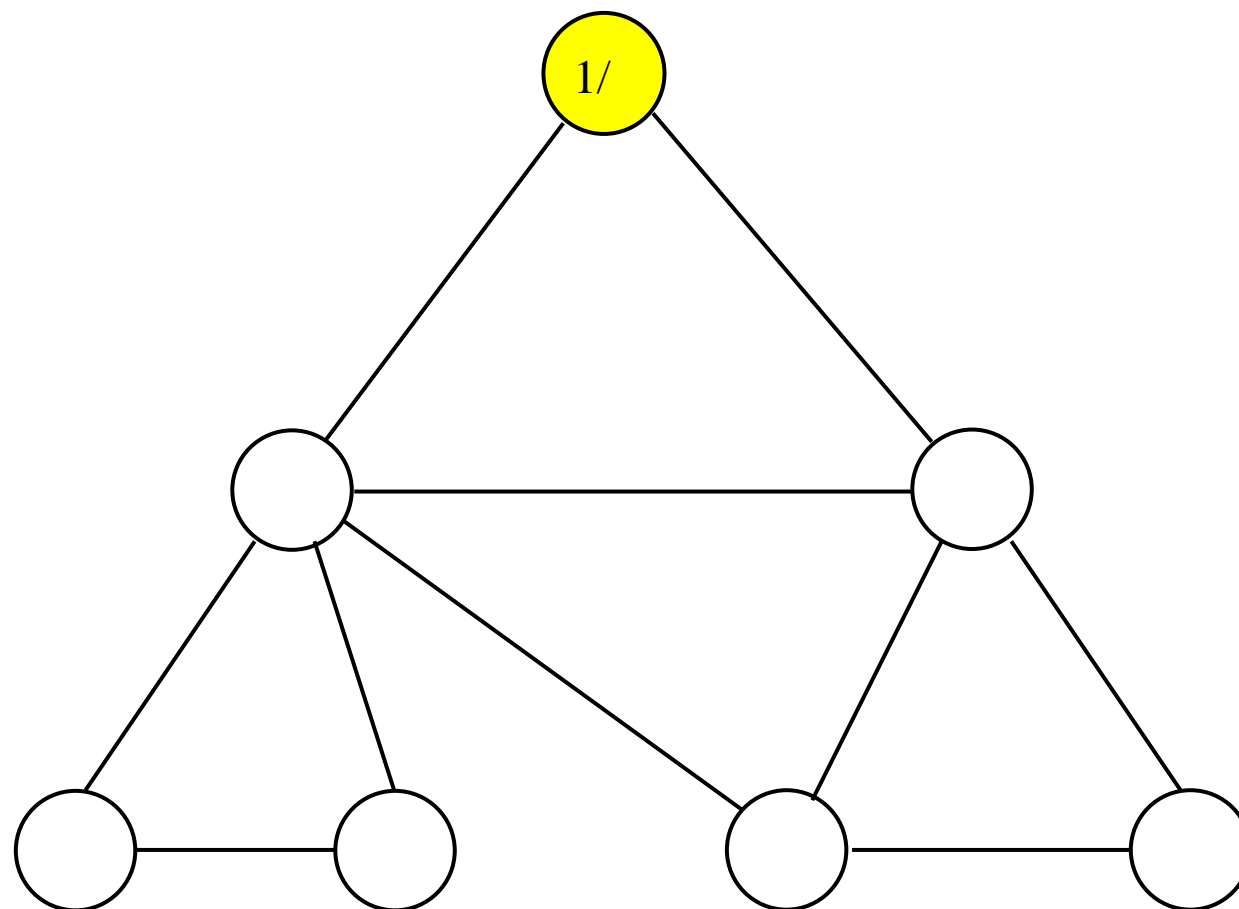
{	do	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		$w \leftarrow I_v[1]$
		$NI_v \leftarrow NI_v - 1$
		for $i \leftarrow 1$ to NI_v
		do $I_v[i] \leftarrow I_v[i - 1]$
		if $Numer[w] = 0$
	{	$Ponumerow \leftarrow Ponumerow + 1$
		$Numer[w] \leftarrow Ponumerow$
		then $Drzewo \leftarrow Drzewo \cup \{vw\}$
		$NStos \leftarrow NStos + 1$
		$Stos[NStos] \leftarrow w$
		else $Pozostałe \leftarrow Pozostałe \cup \{vw\}$
return ($Numer, Drzewo, Pozostałe$)		

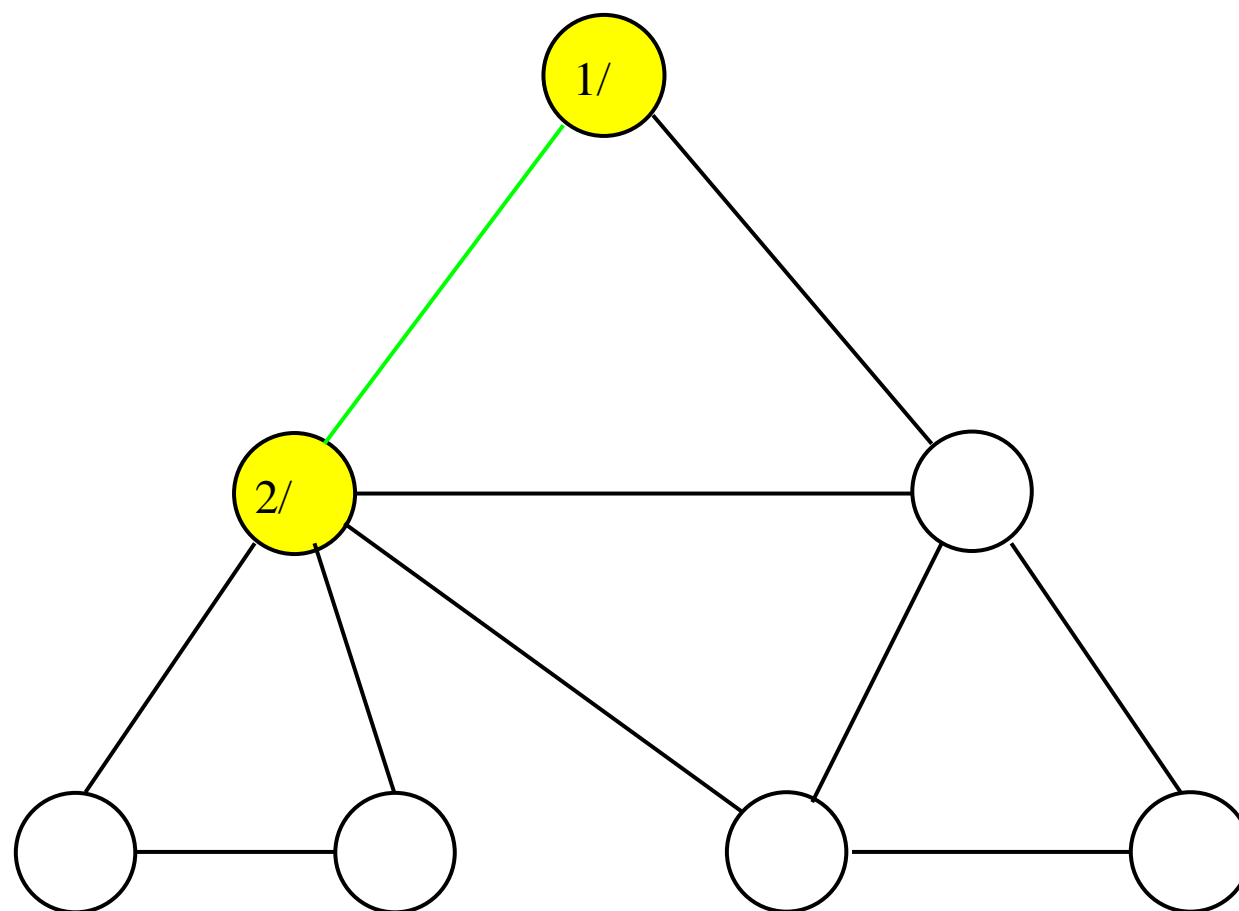
14 Próba wyjścia z wierzchołka a  $v \leftarrow a$ $w \leftarrow c$ $NStos \leftarrow 0$ $Stos \leftarrow []$

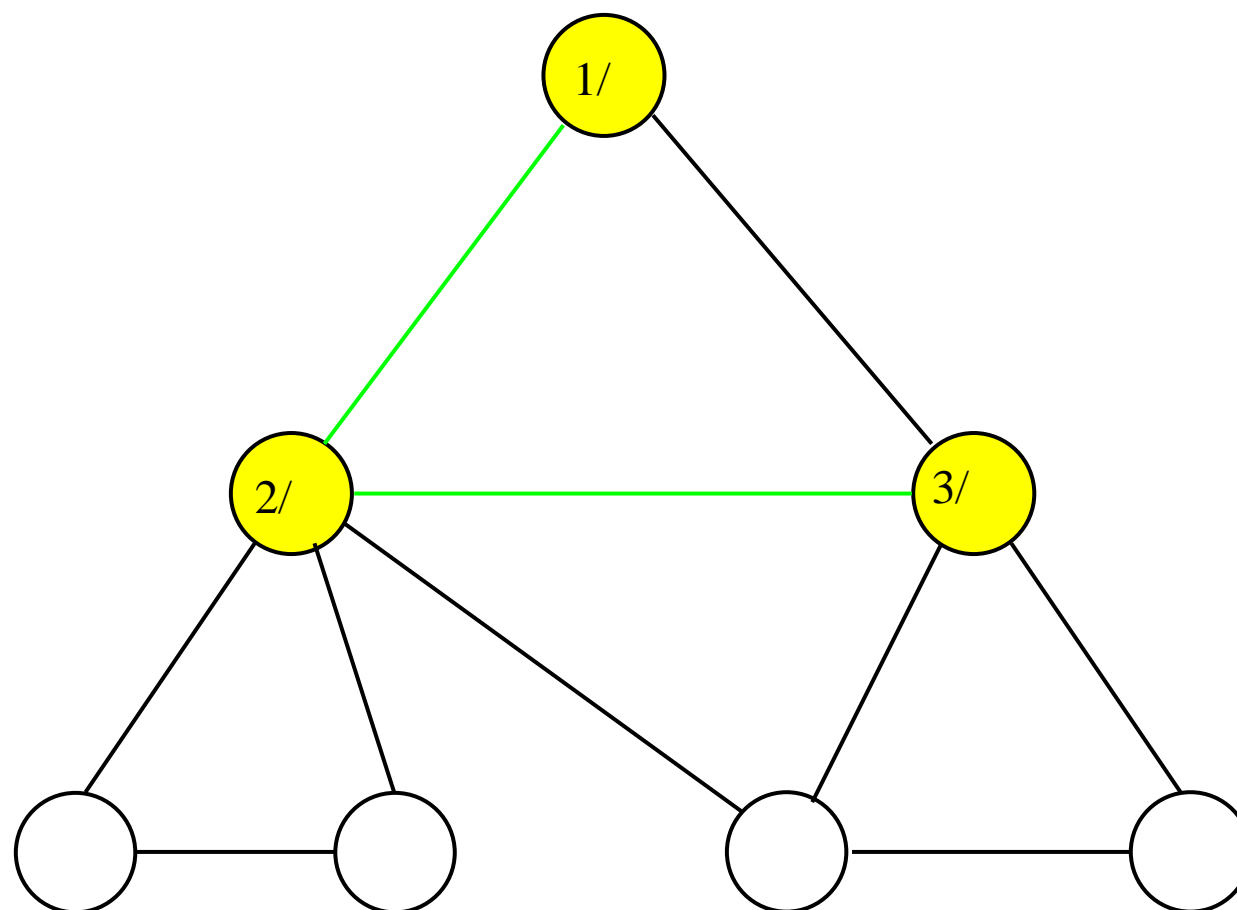
Własności DFS

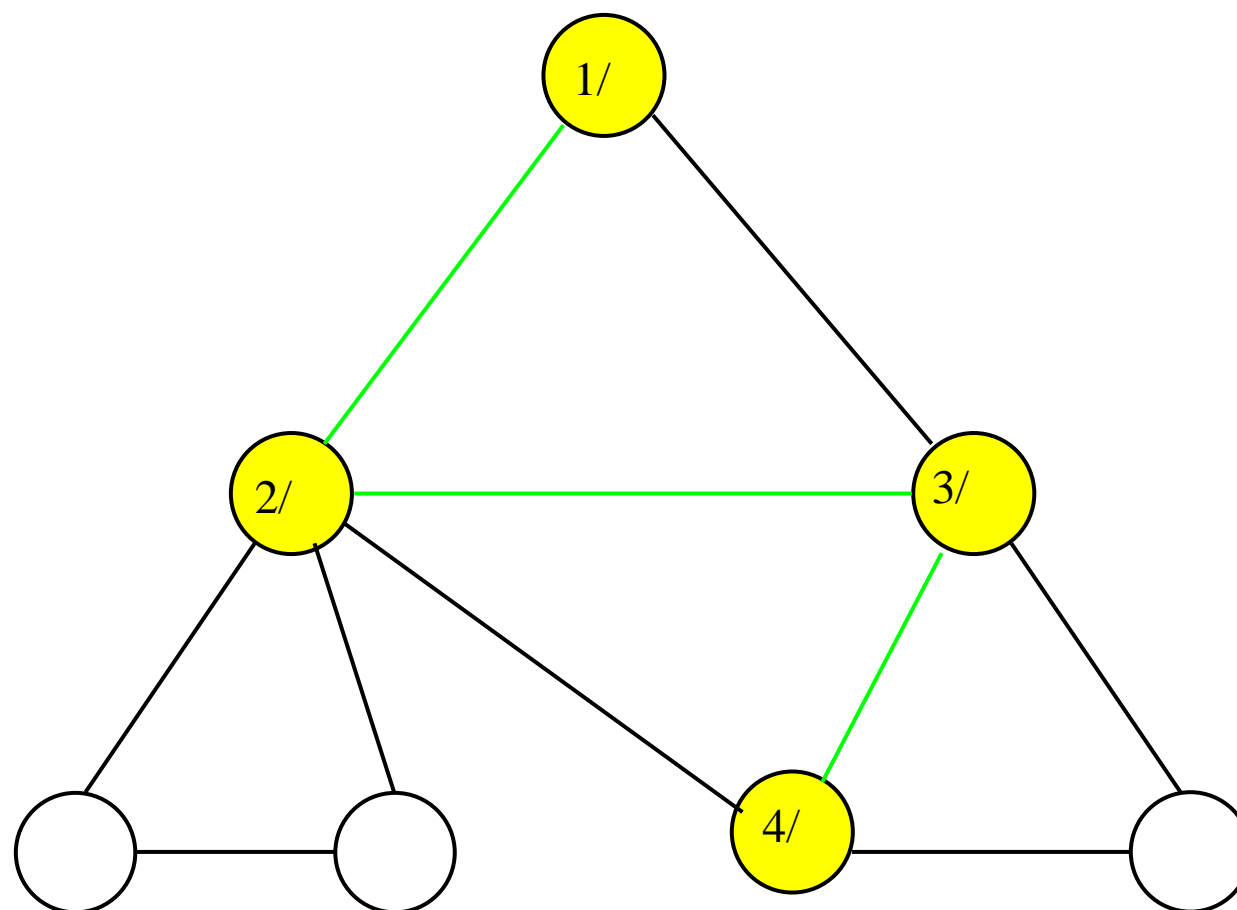
- las przeszukiwań w głąb (drzewo gdy graf jest spójny)
- wierzchołek **odwiedzony** - odwiedzony po raz pierwszy podczas przeszukiwania
- wierzchołek **przetworzony** - lista sąsiedztwa została całkowicie zbadana
- etykiety czasowe : każdemu wierzchołki przyporządkowujemy dwie liczby całkowite z przedziału $1, \dots, 2|V|$
 - czas odwiedzin
 - czas przetworzenia

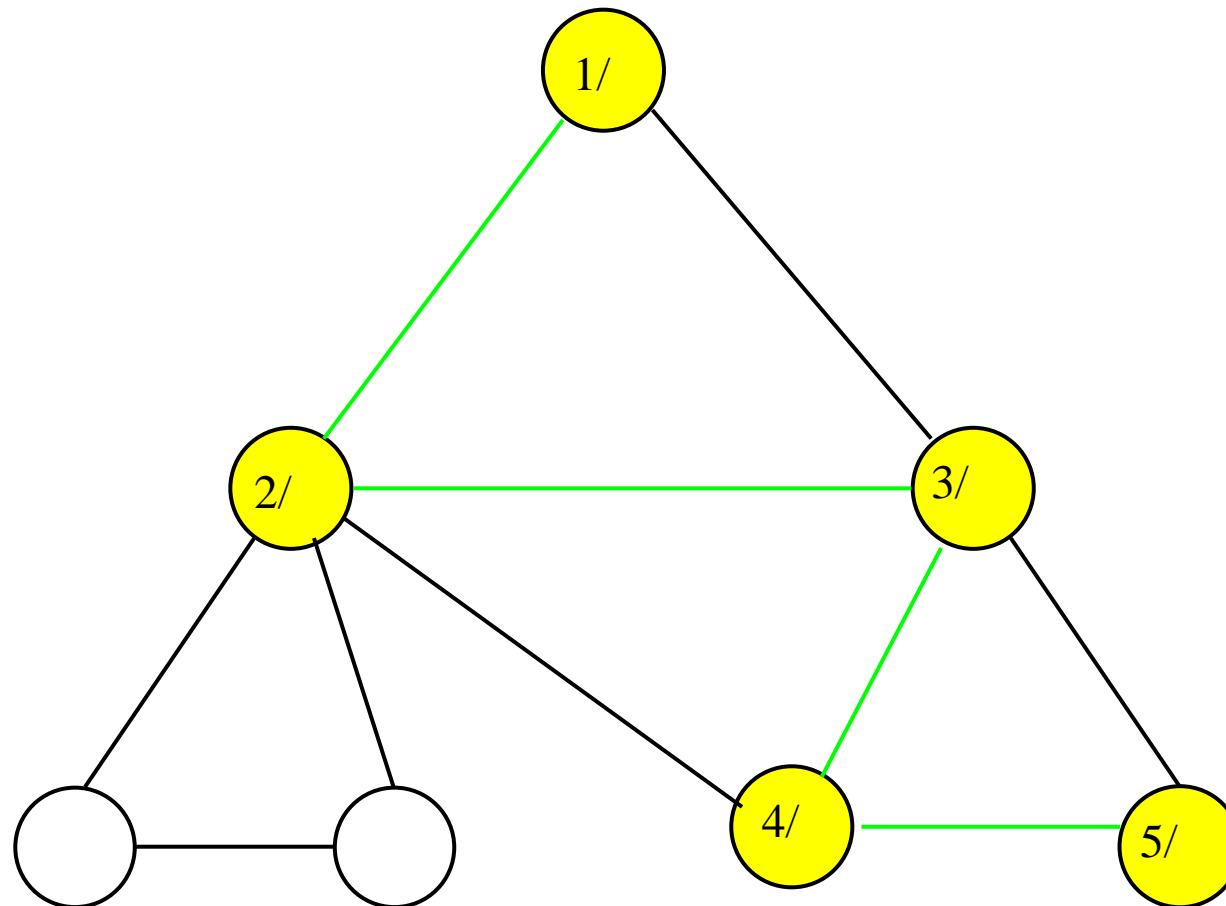


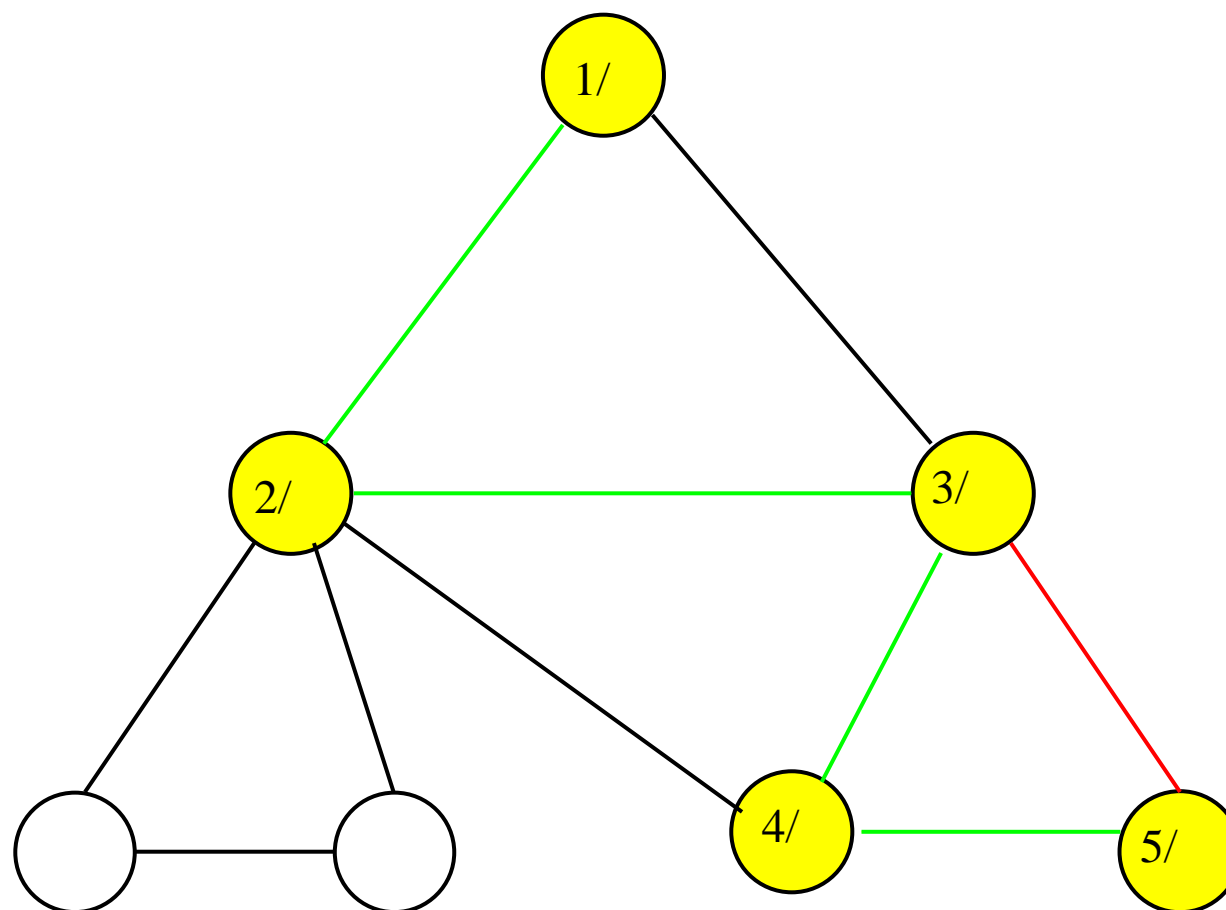


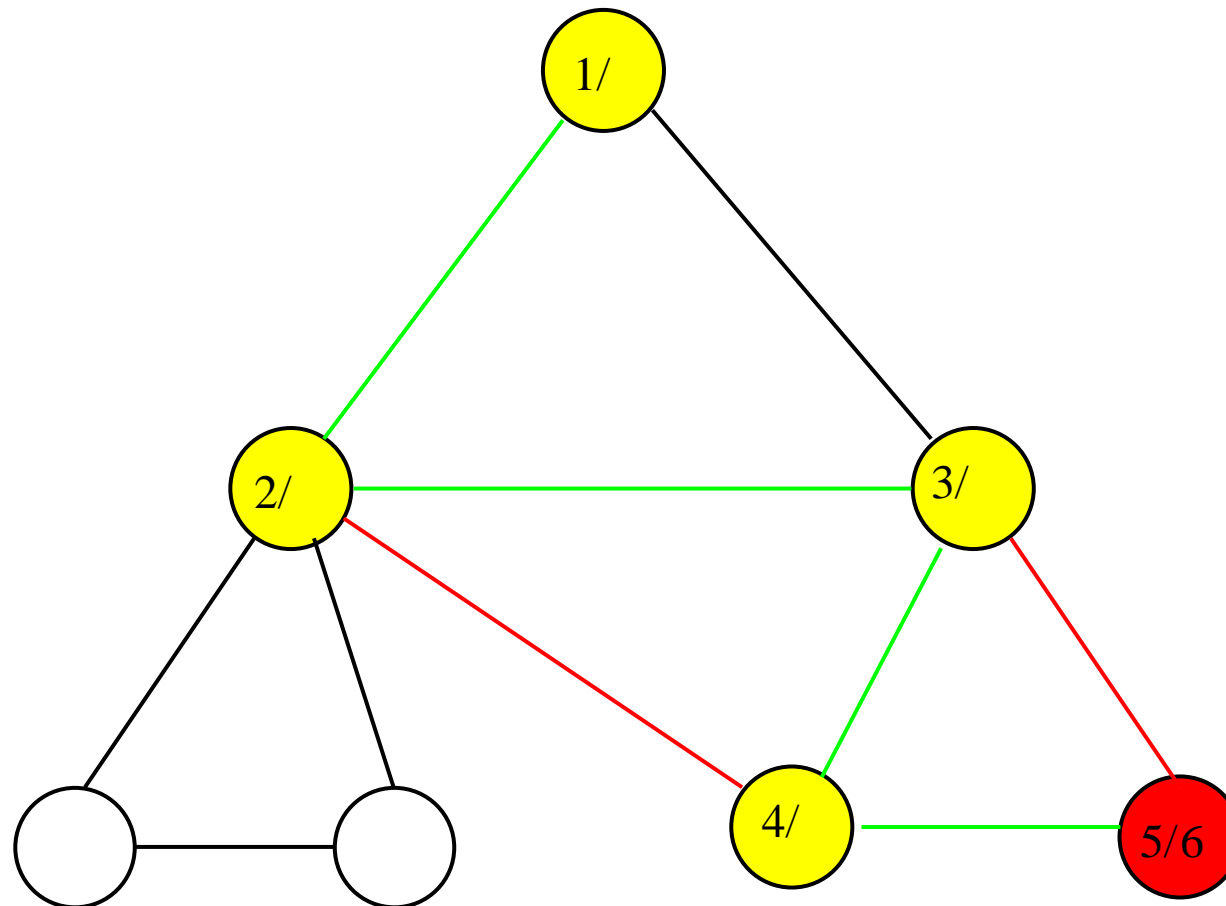


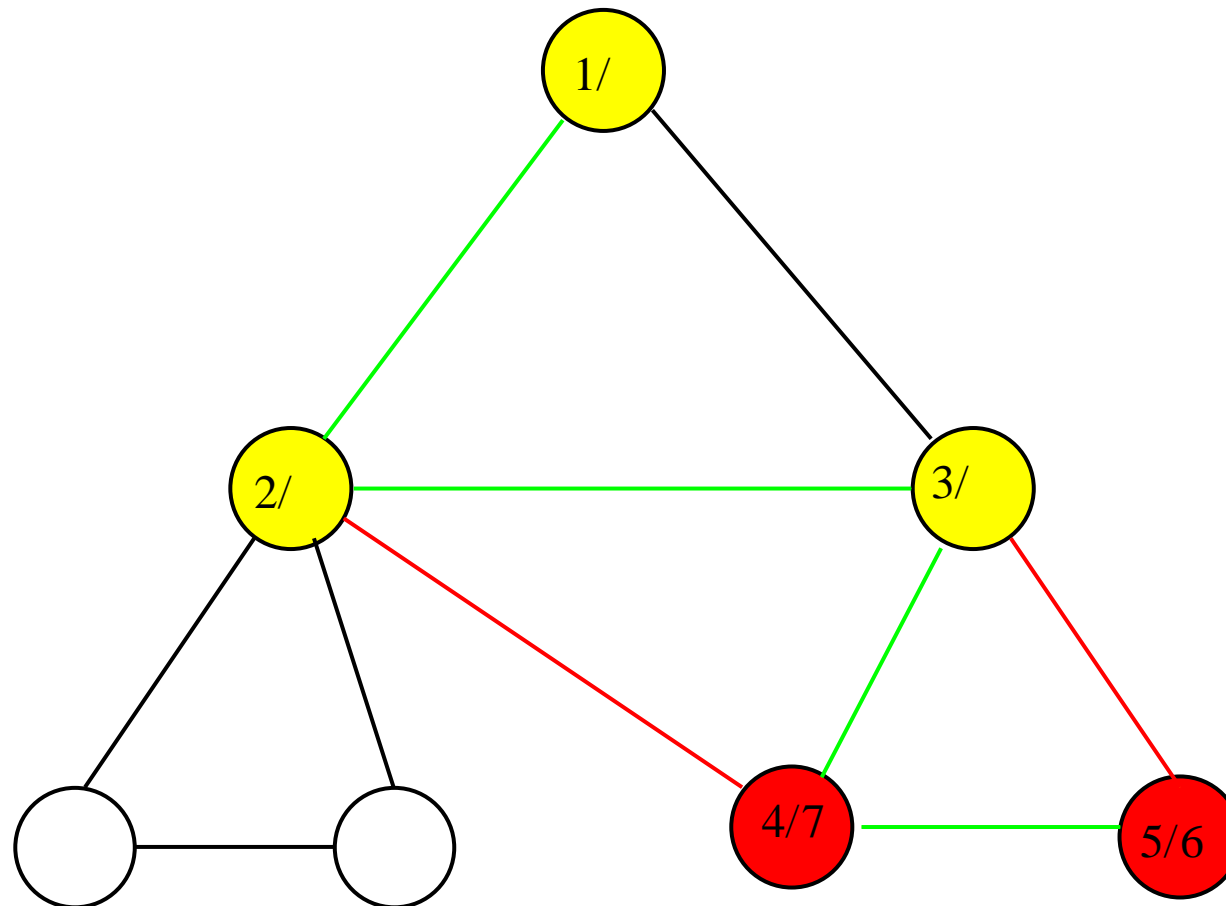


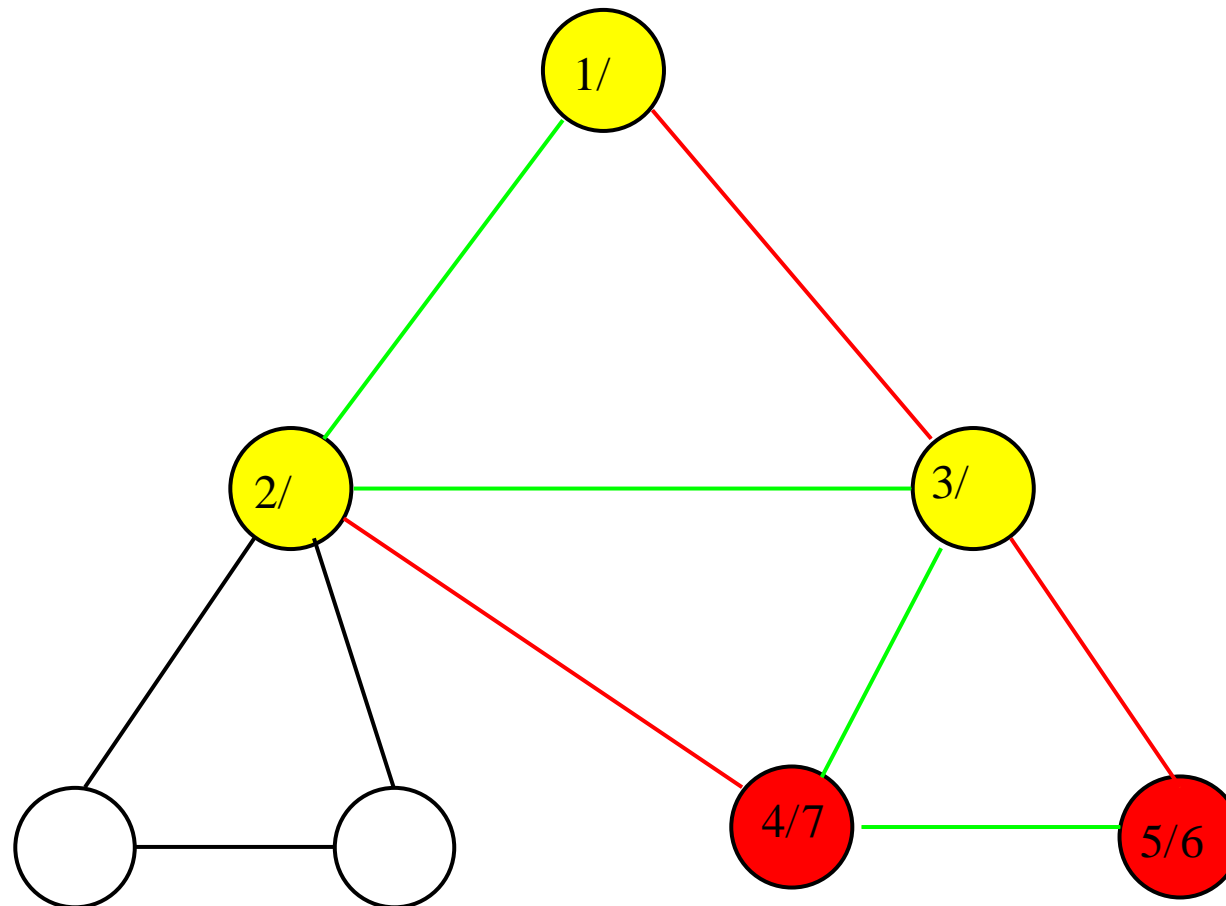


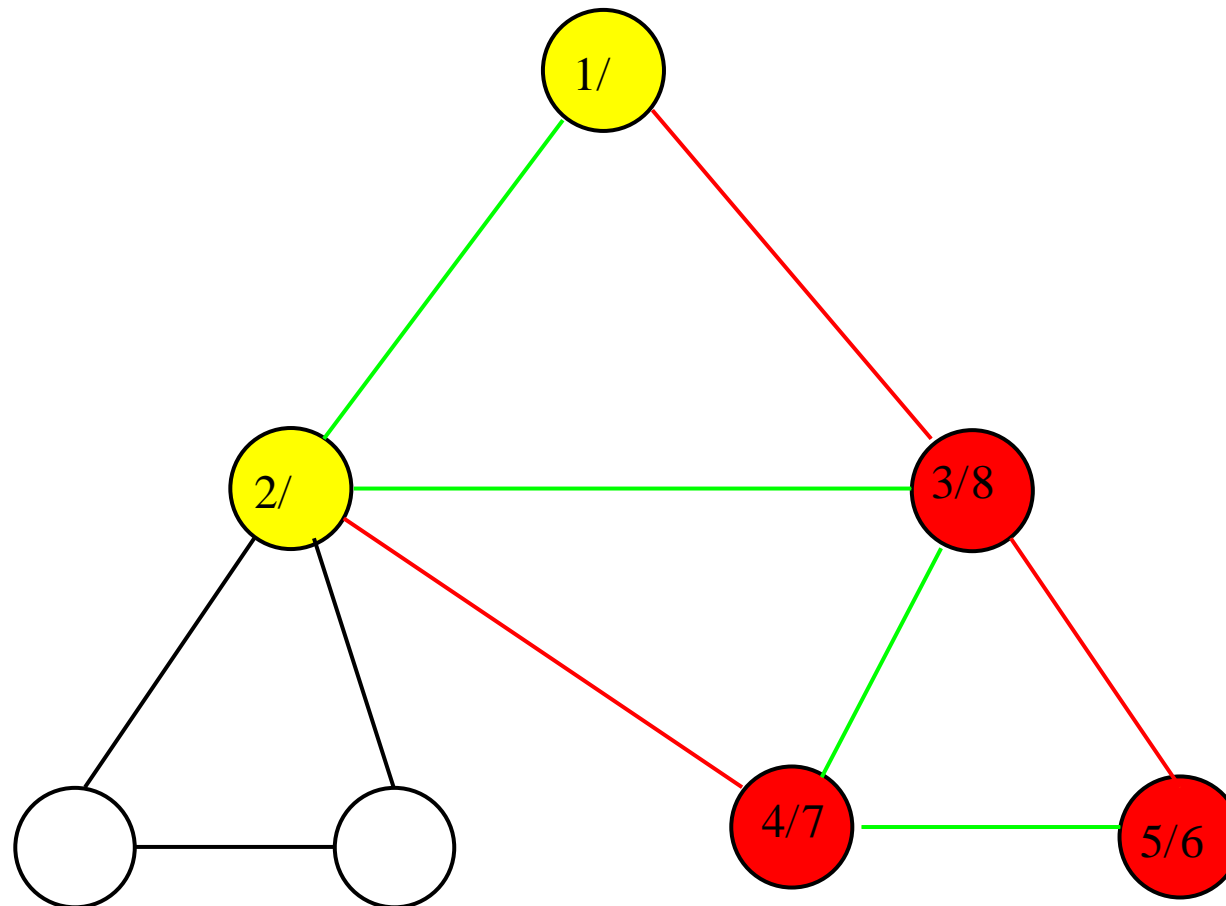


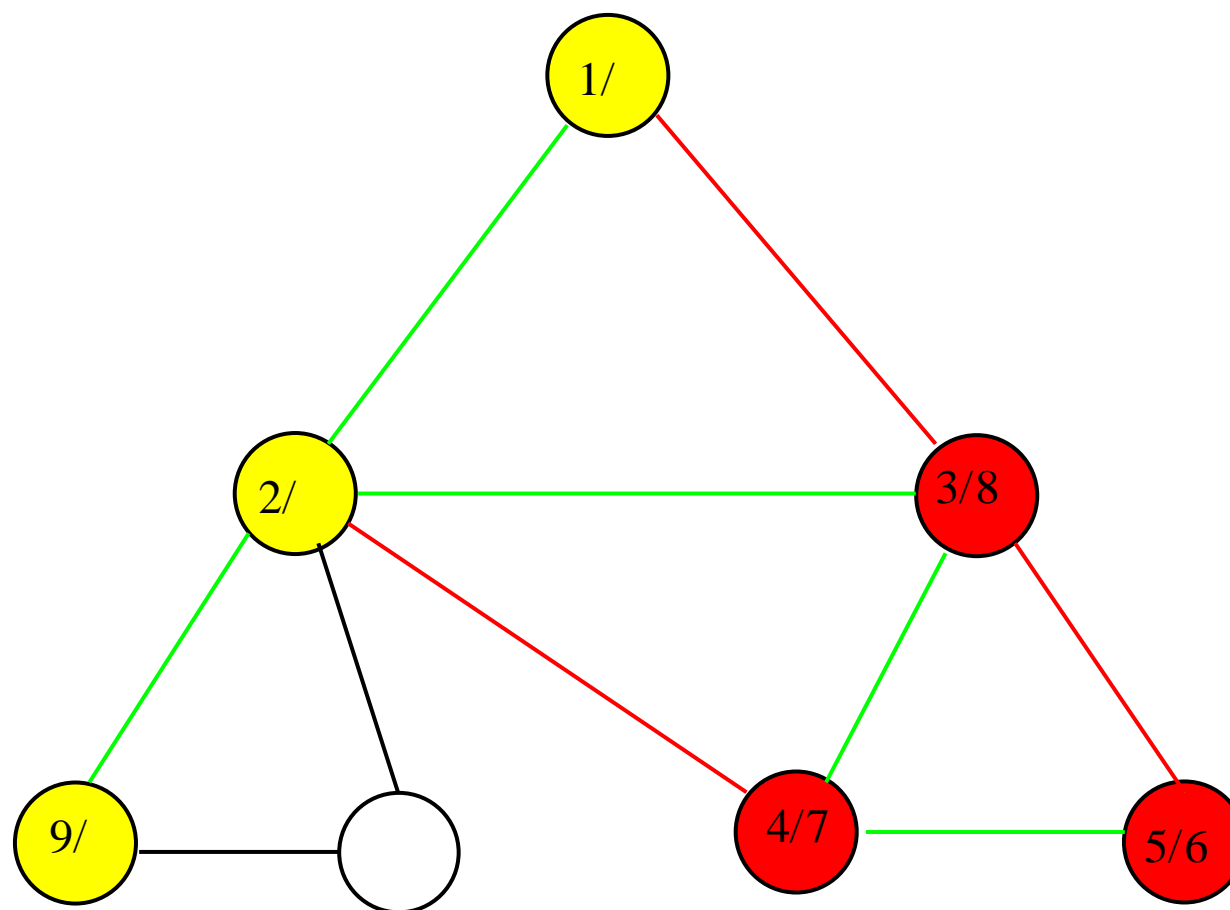


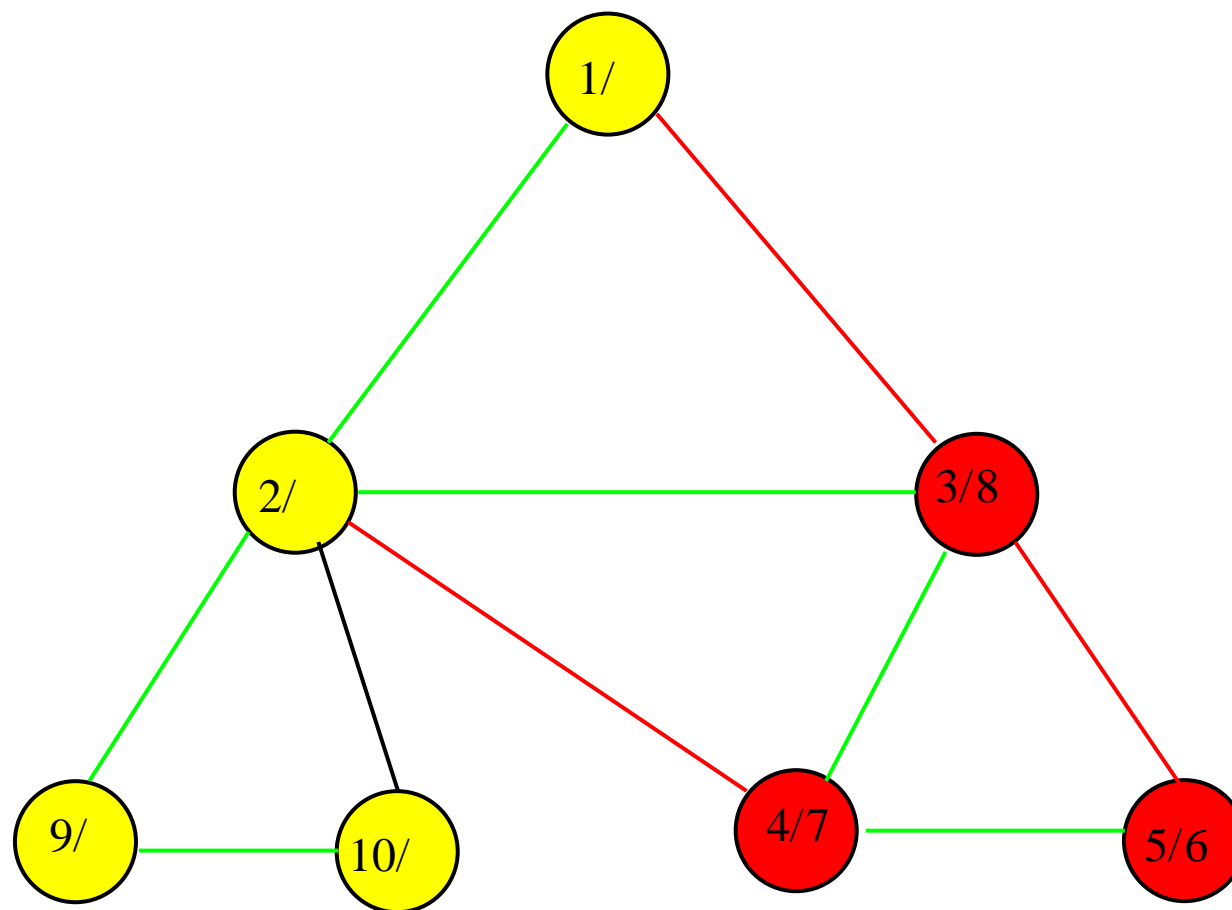


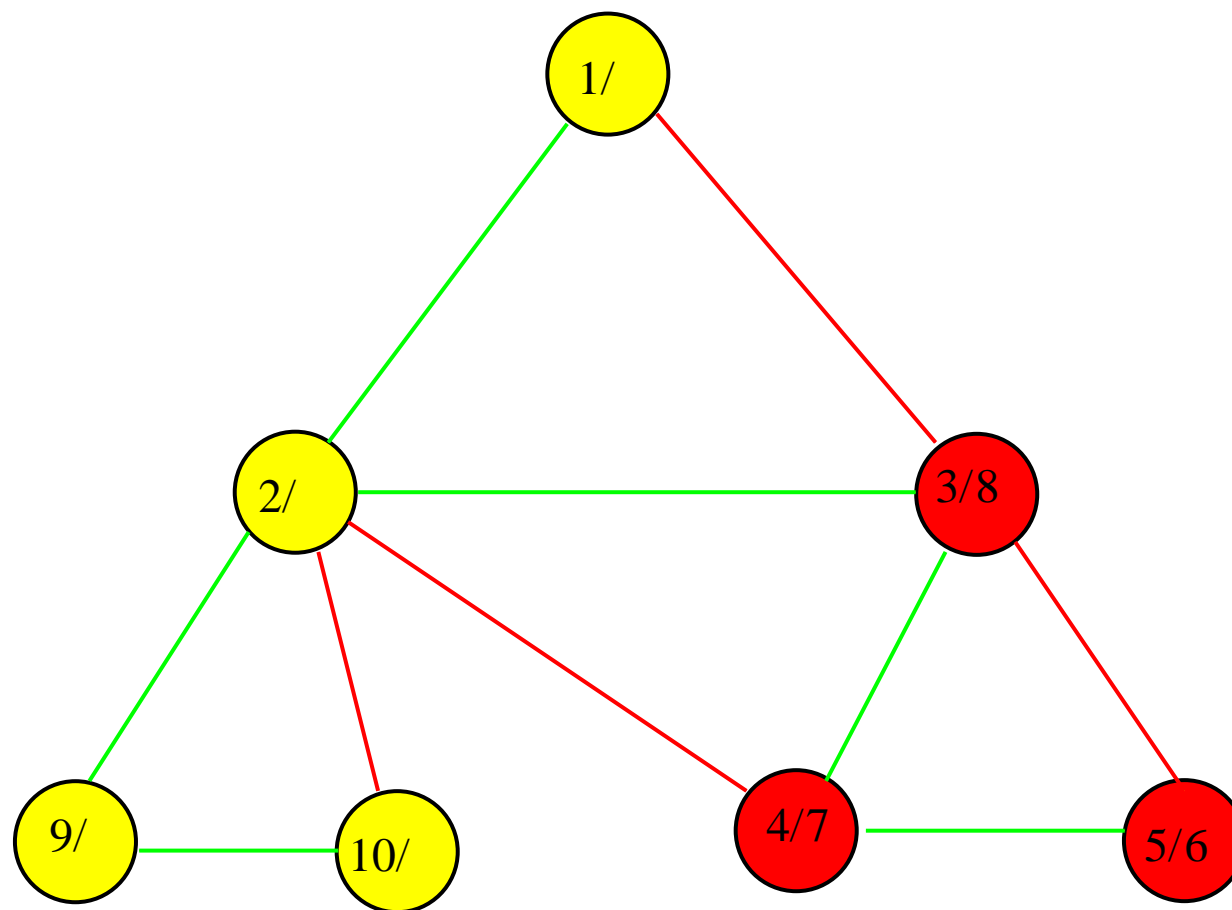


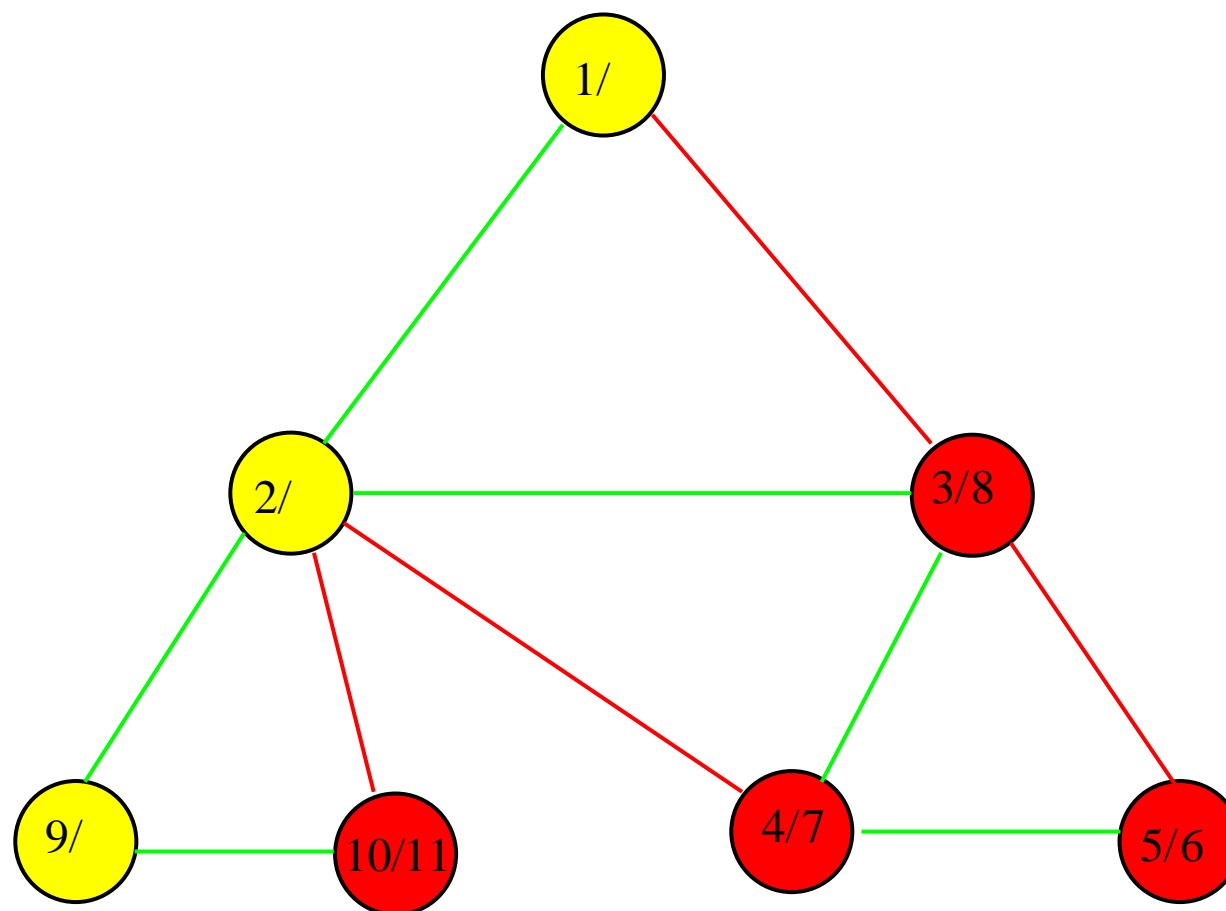


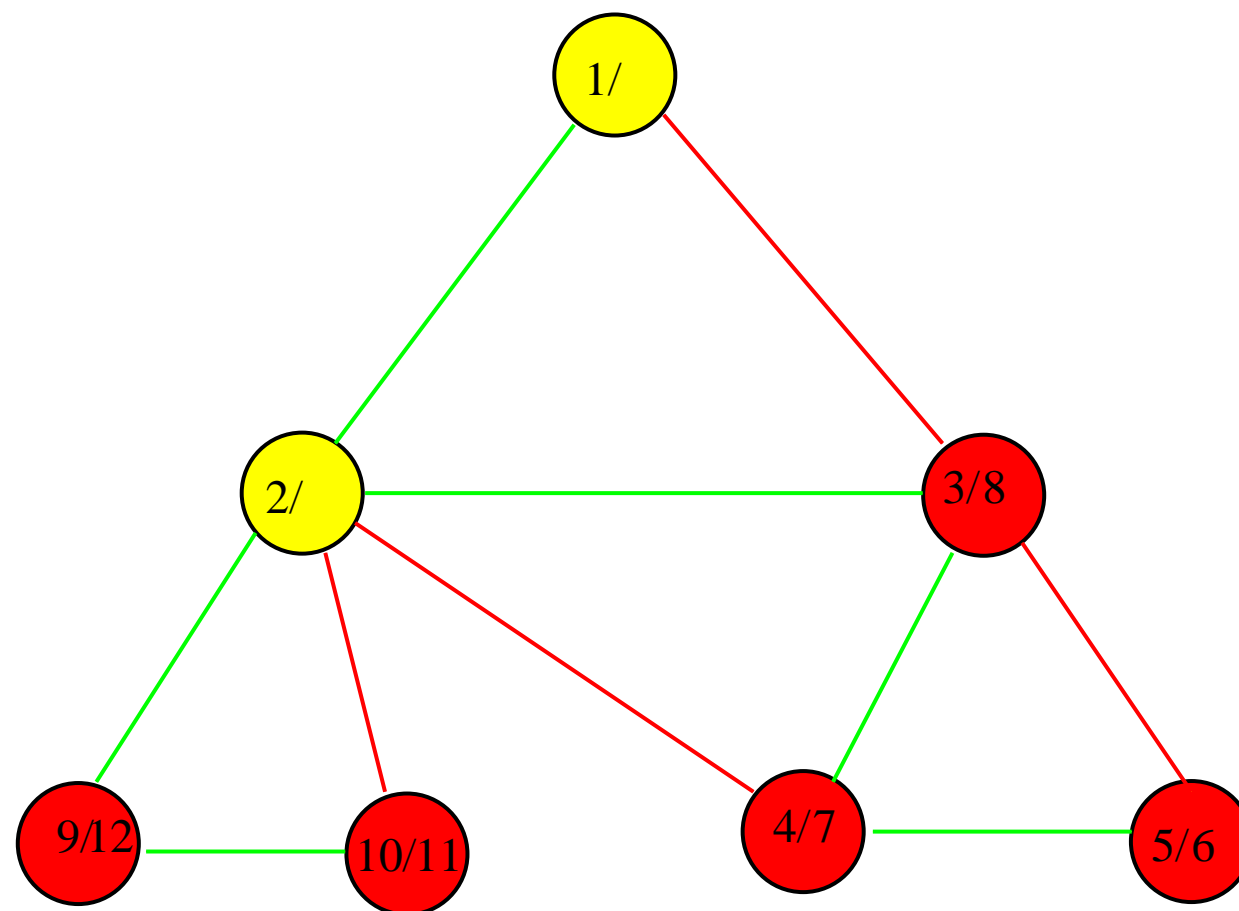


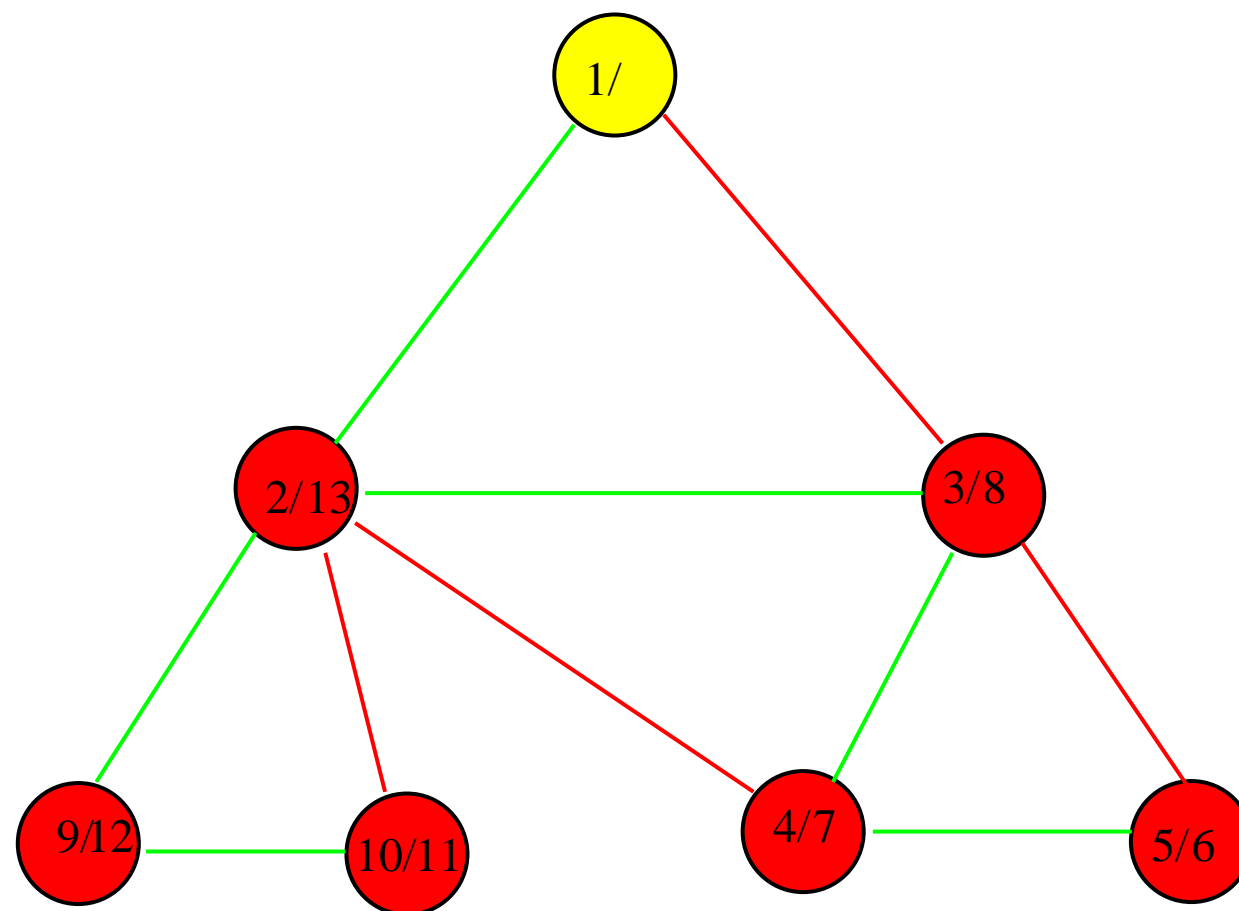


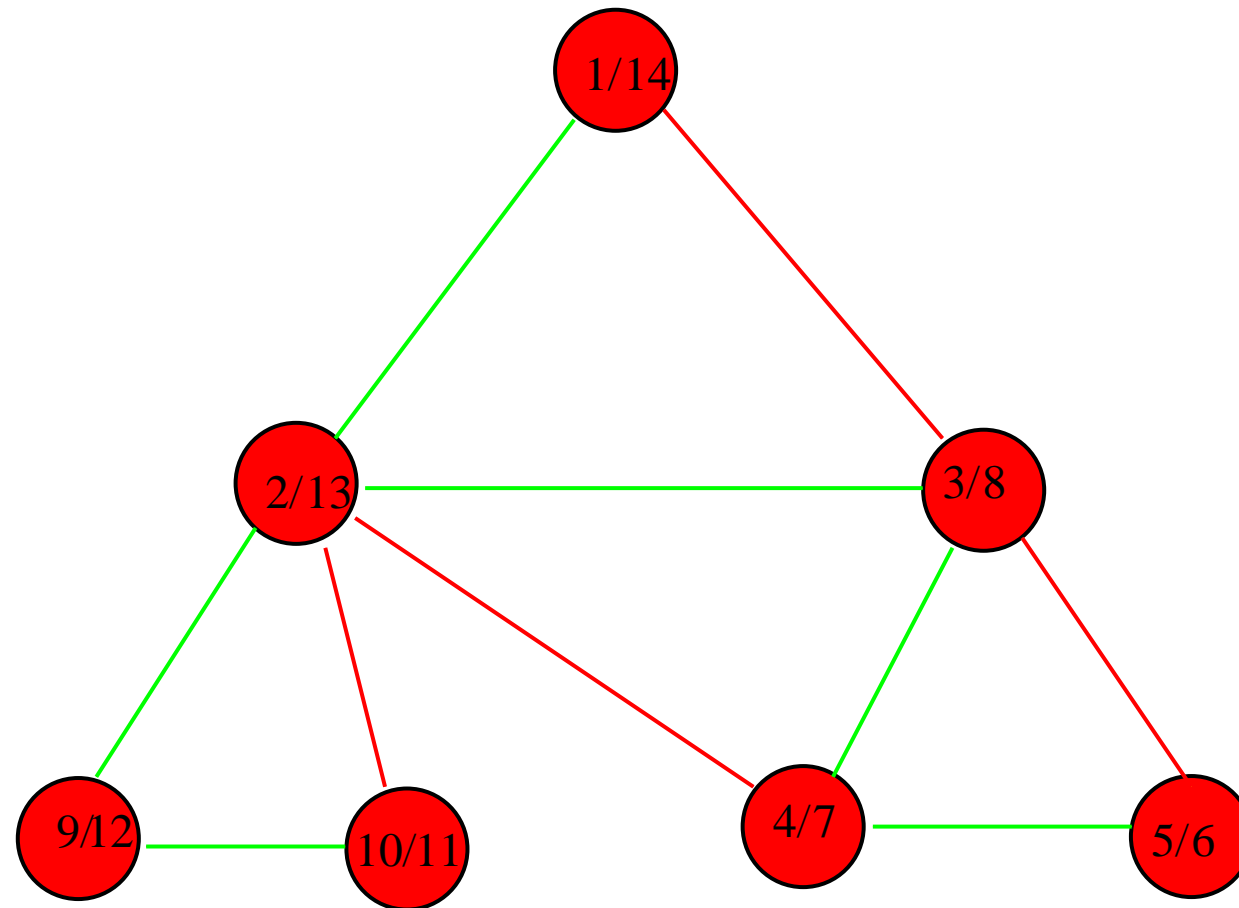


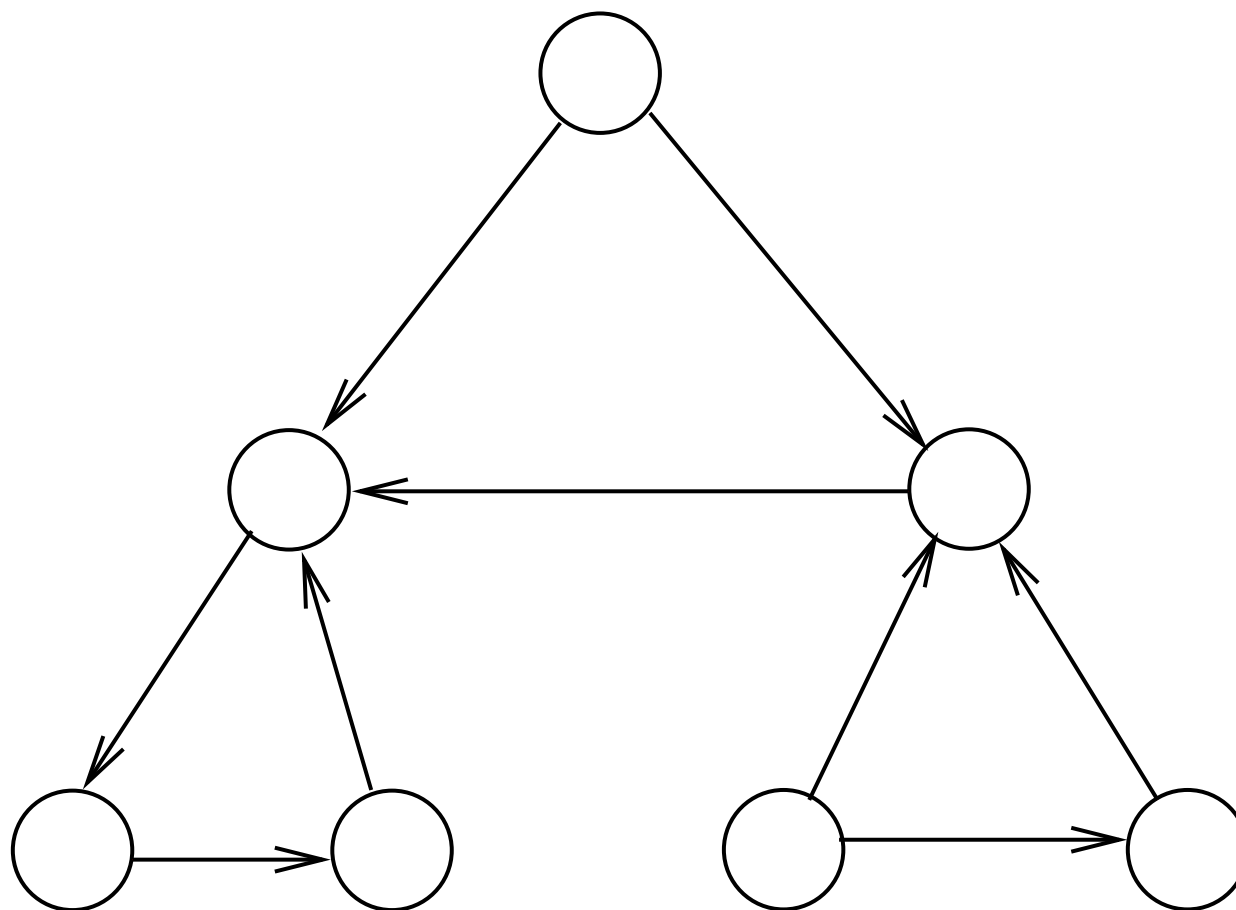


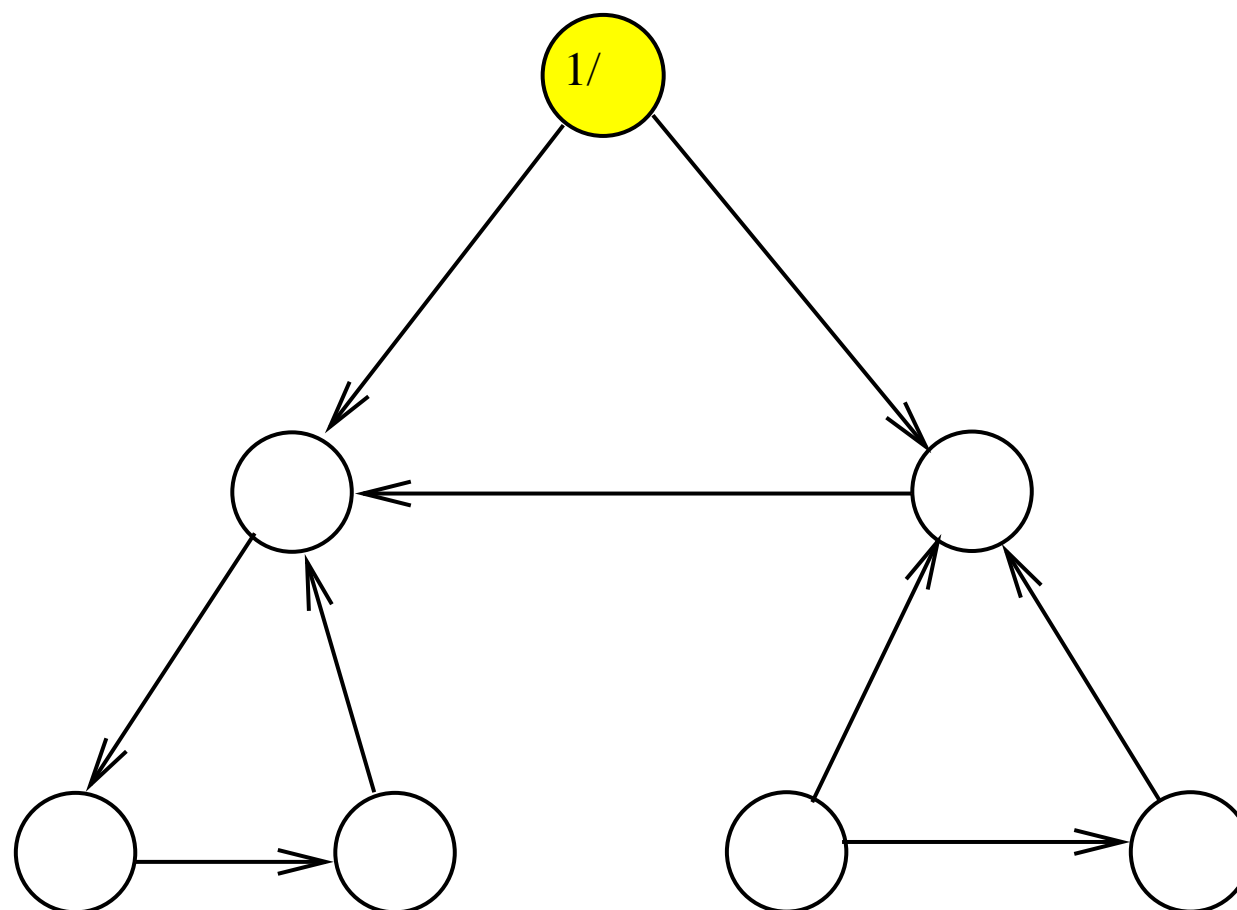


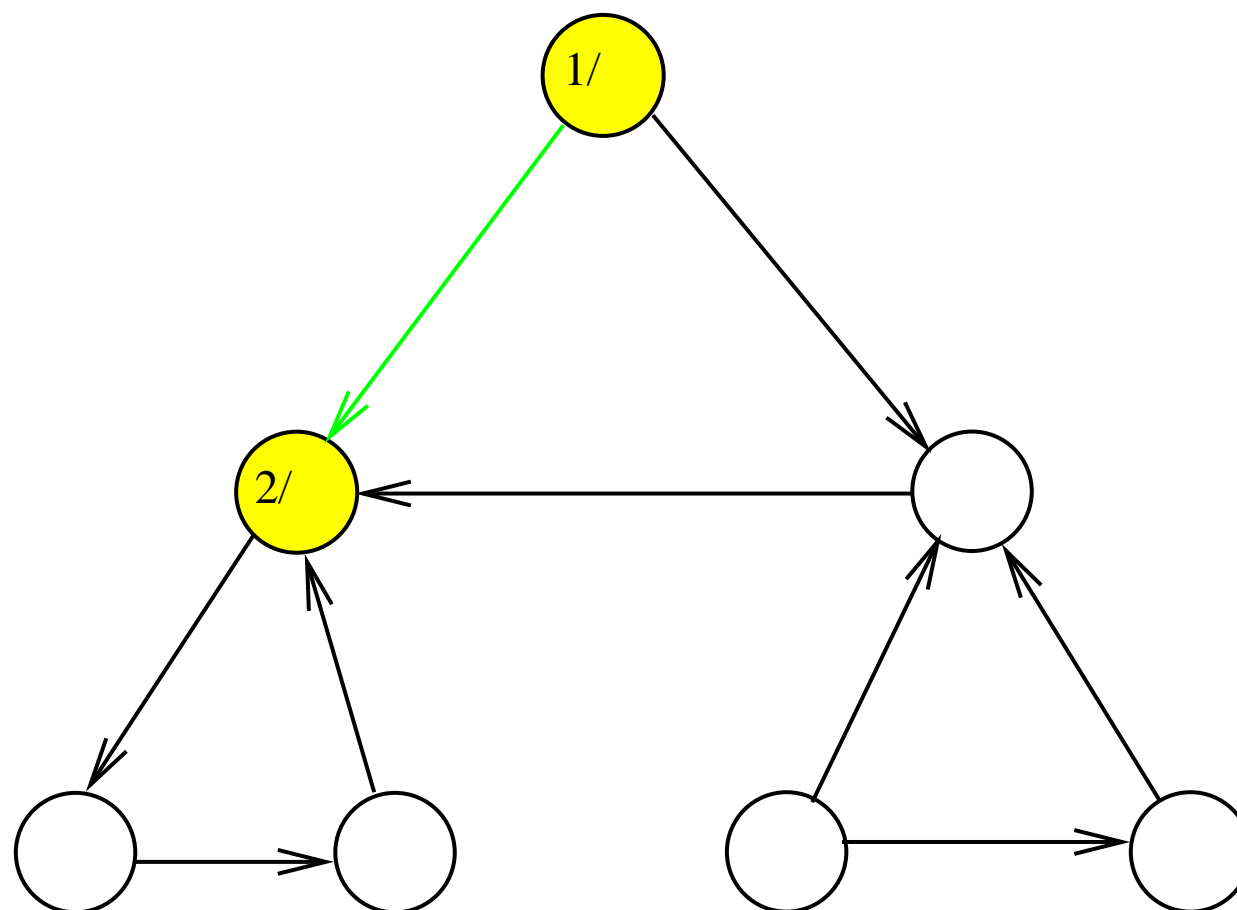


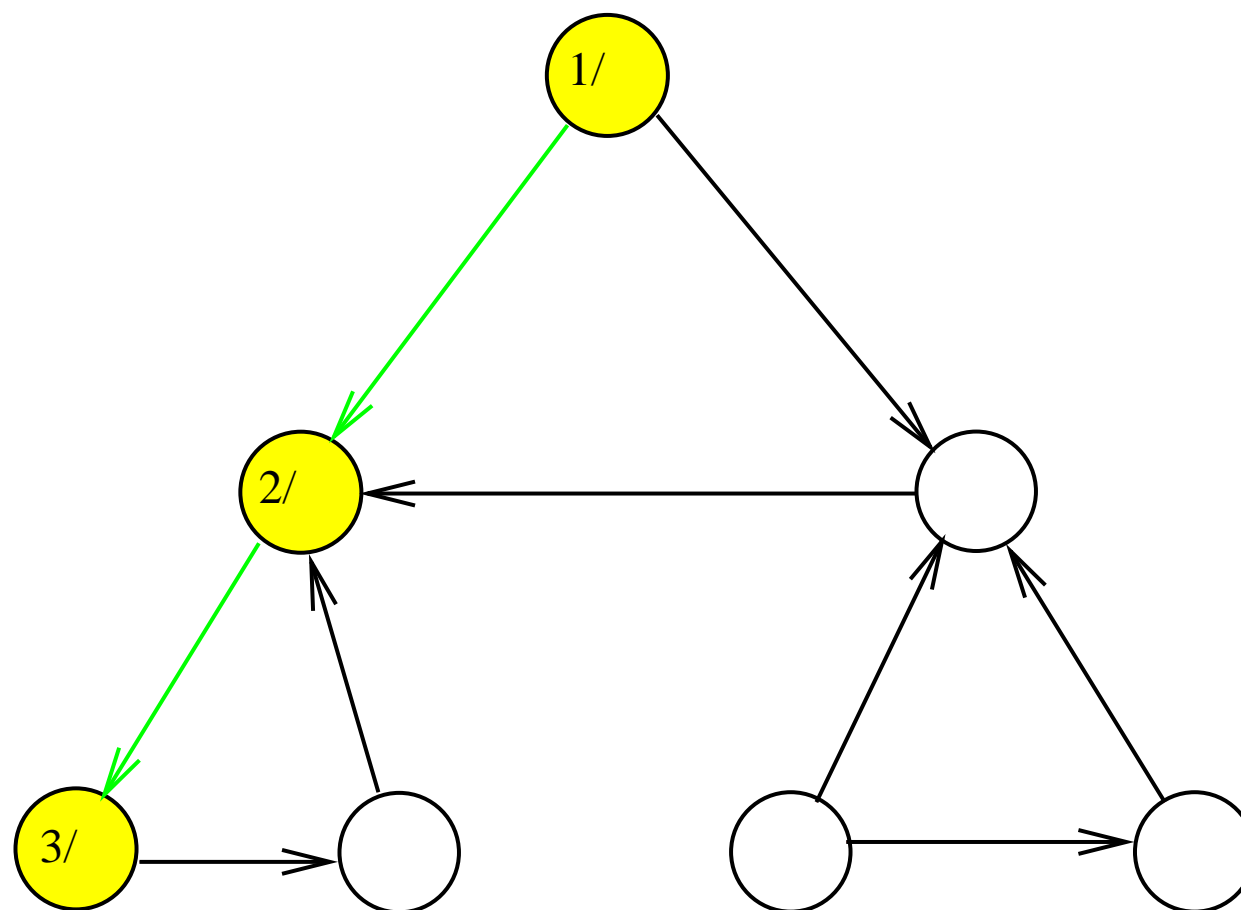


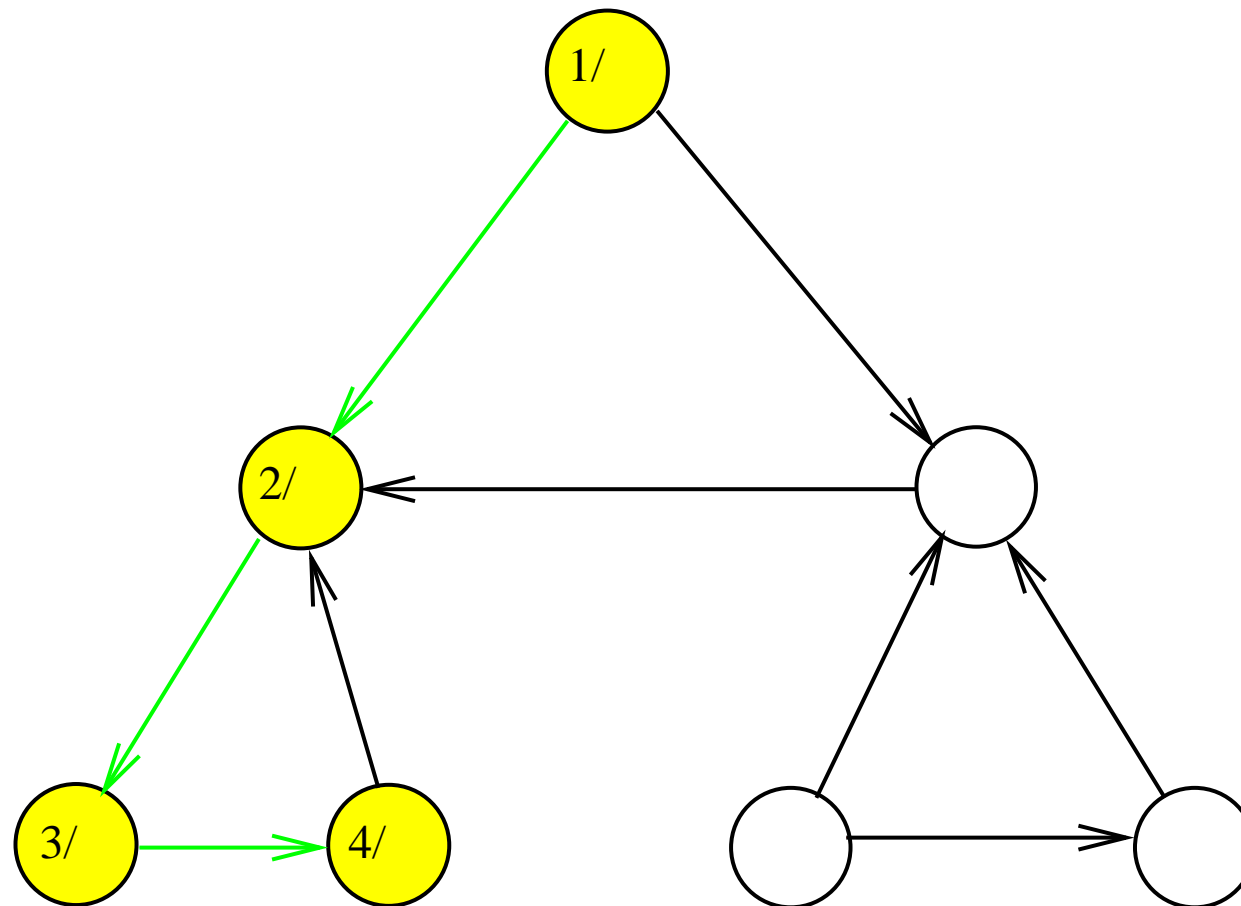


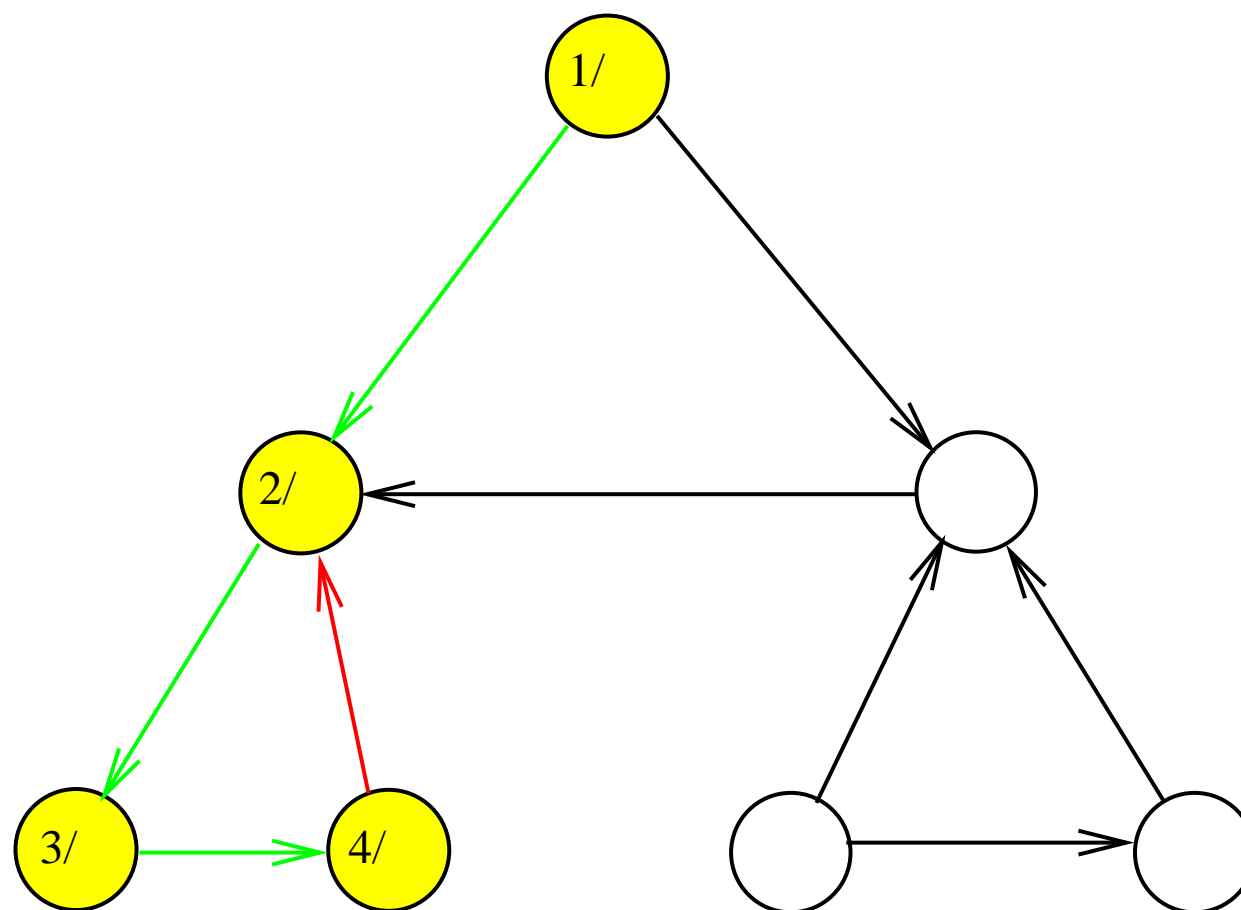


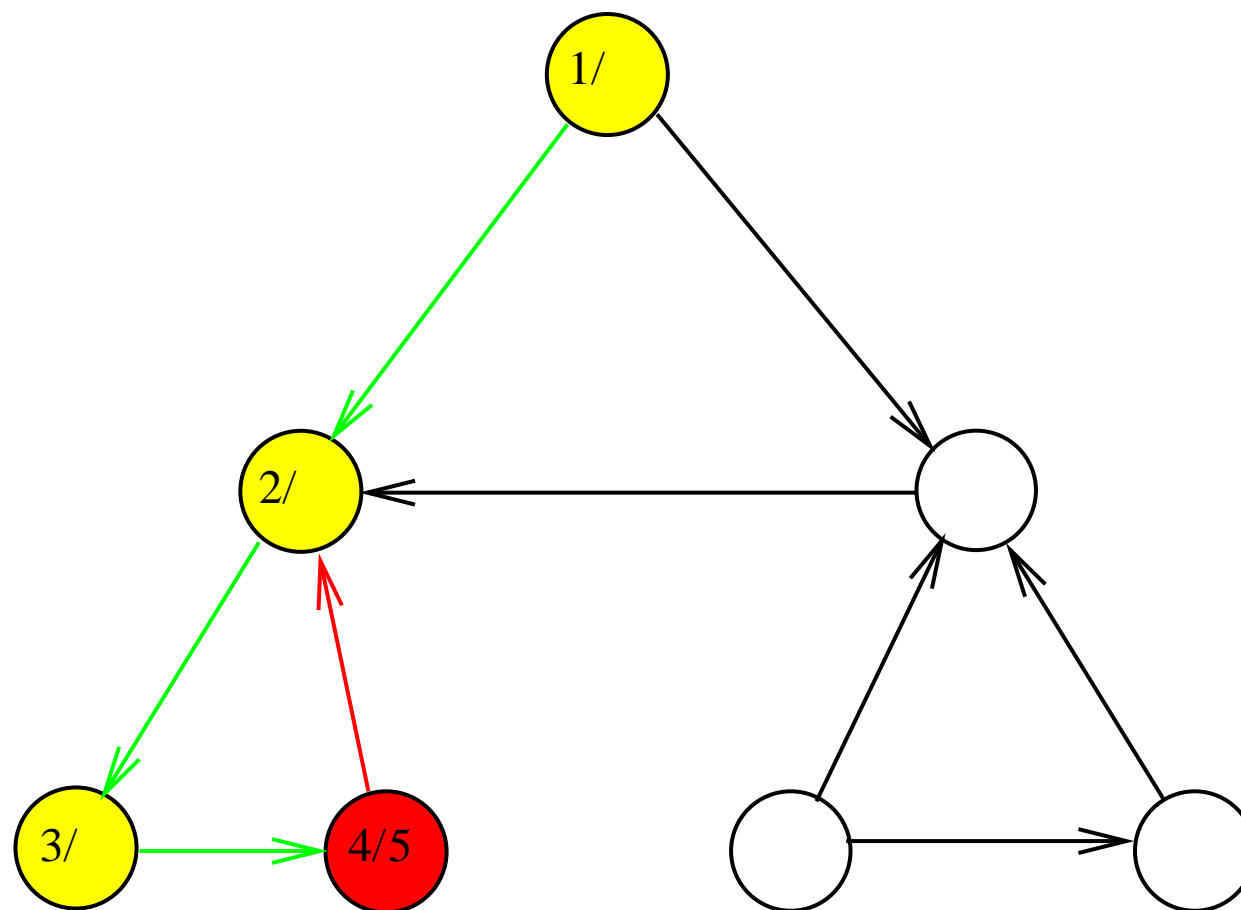


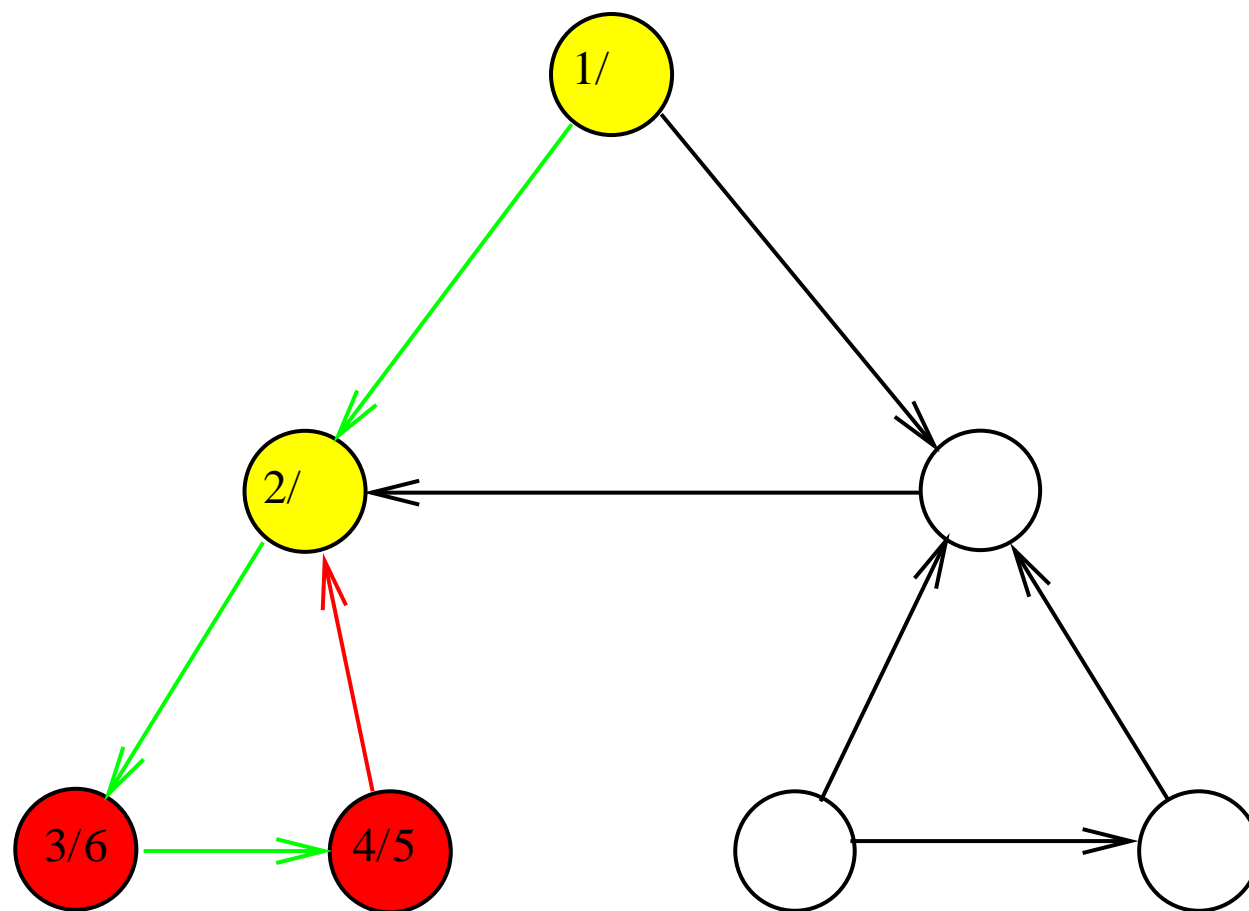


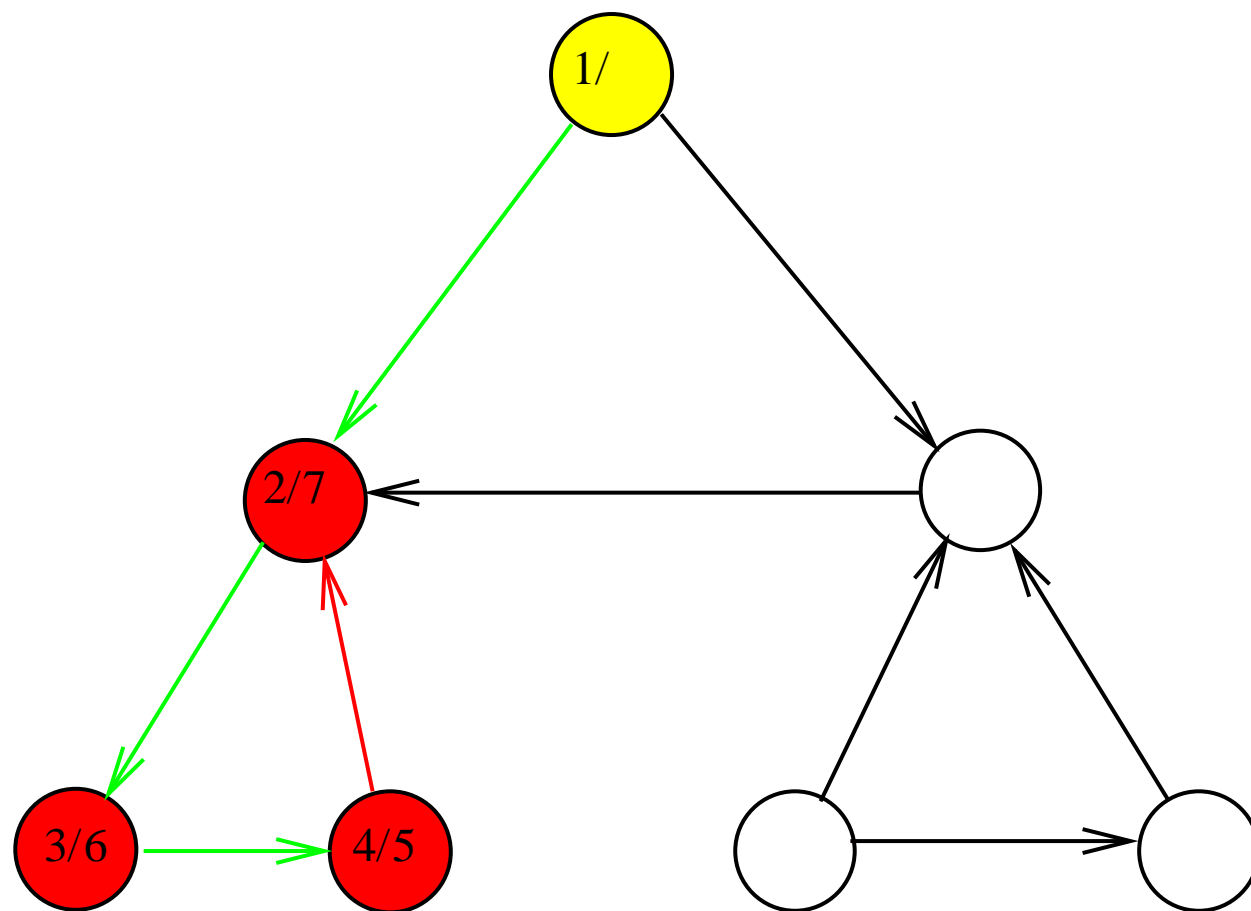


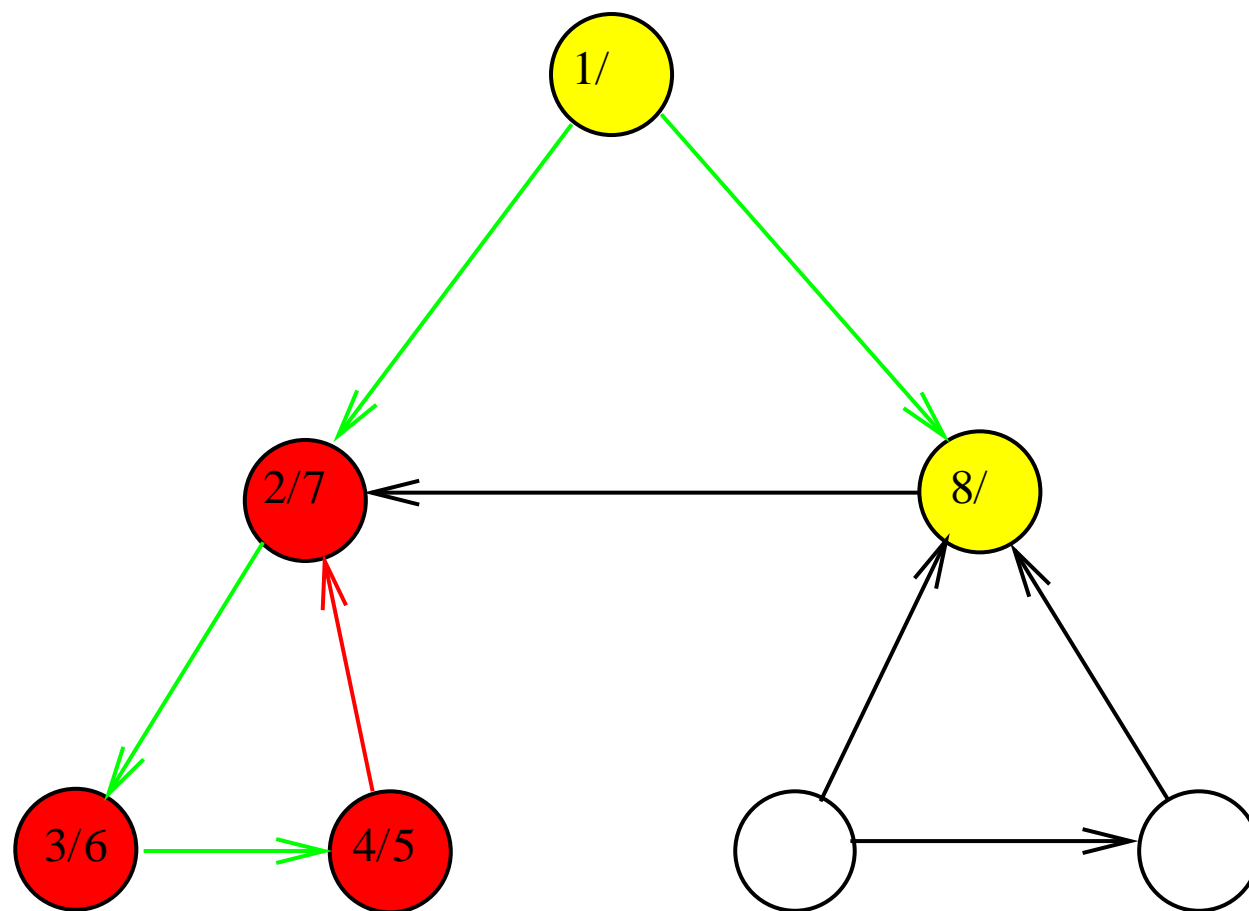


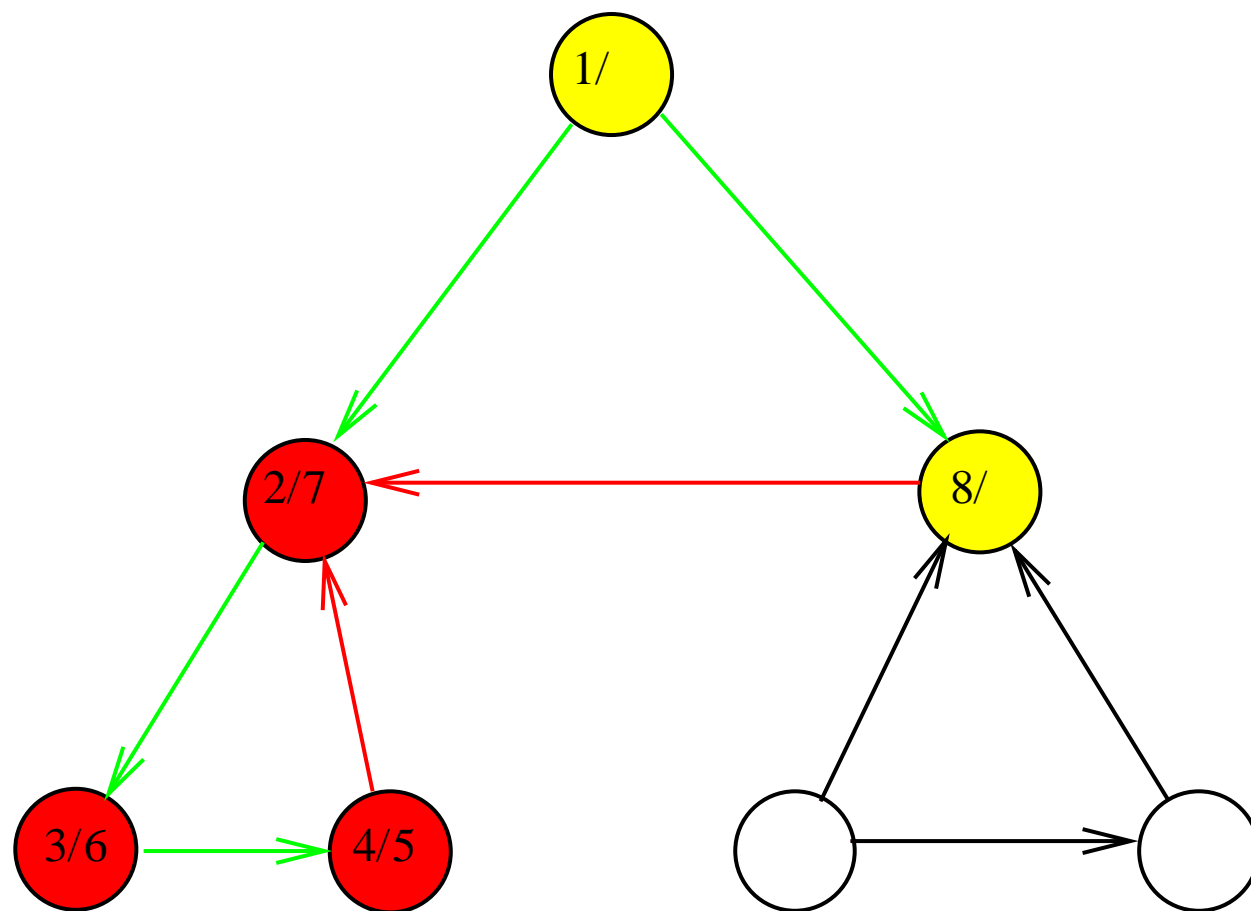


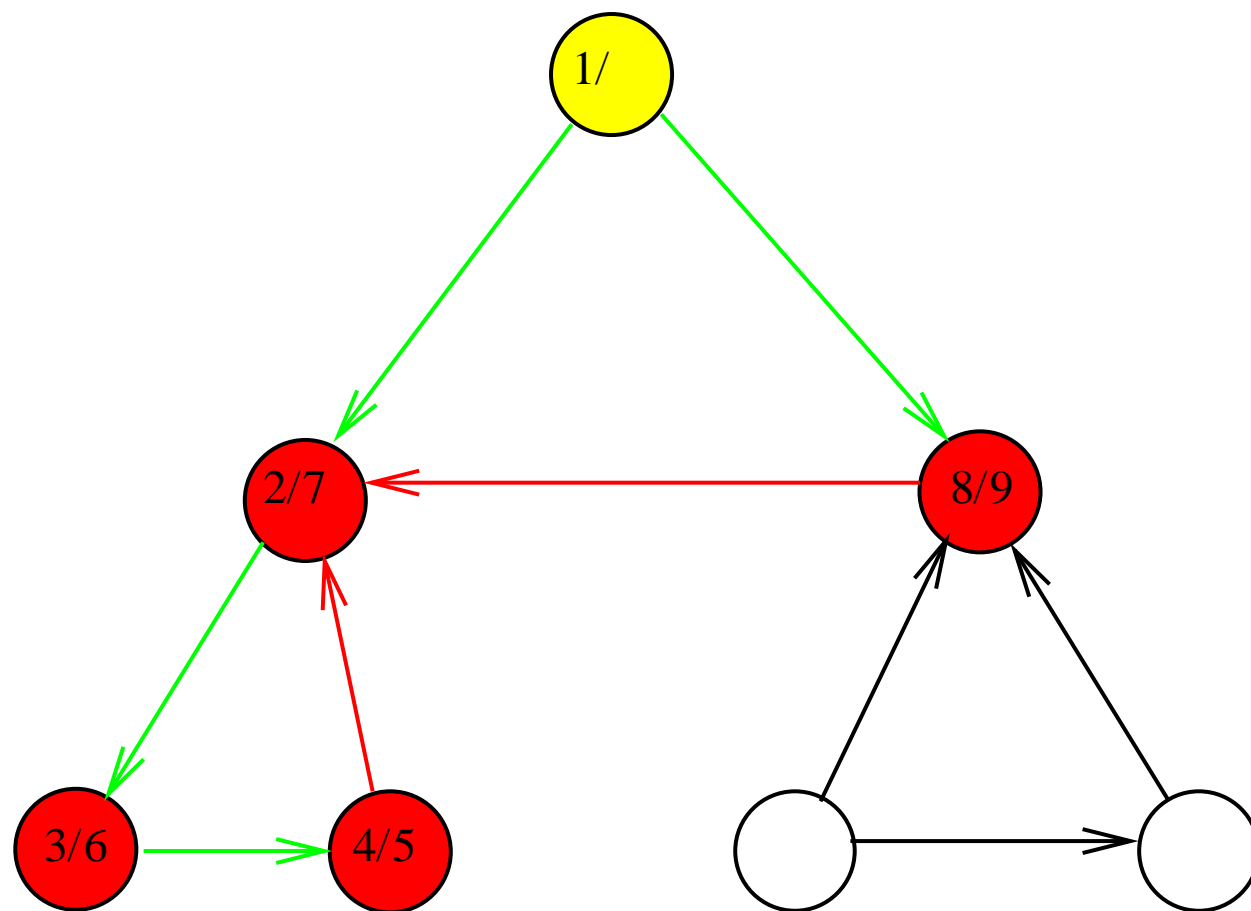


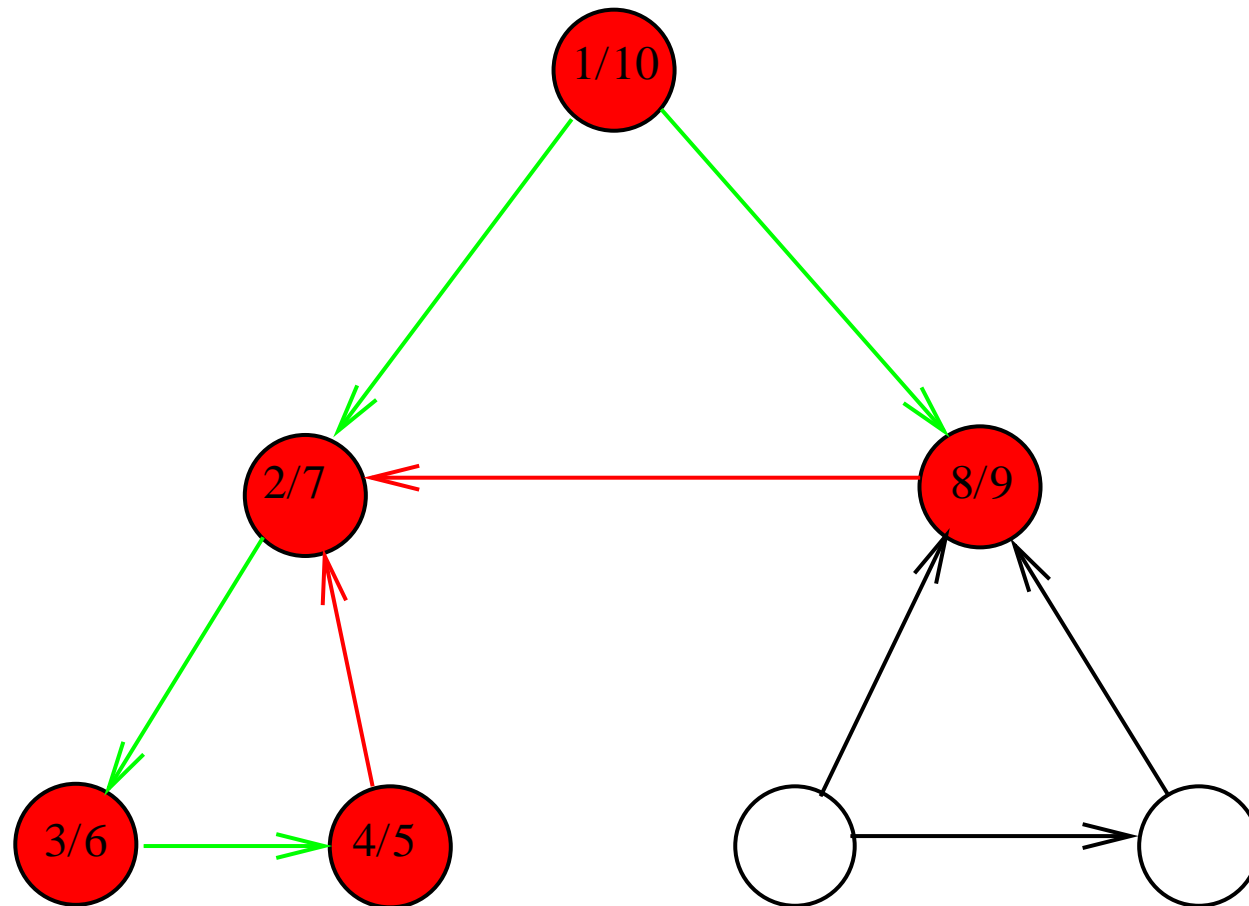


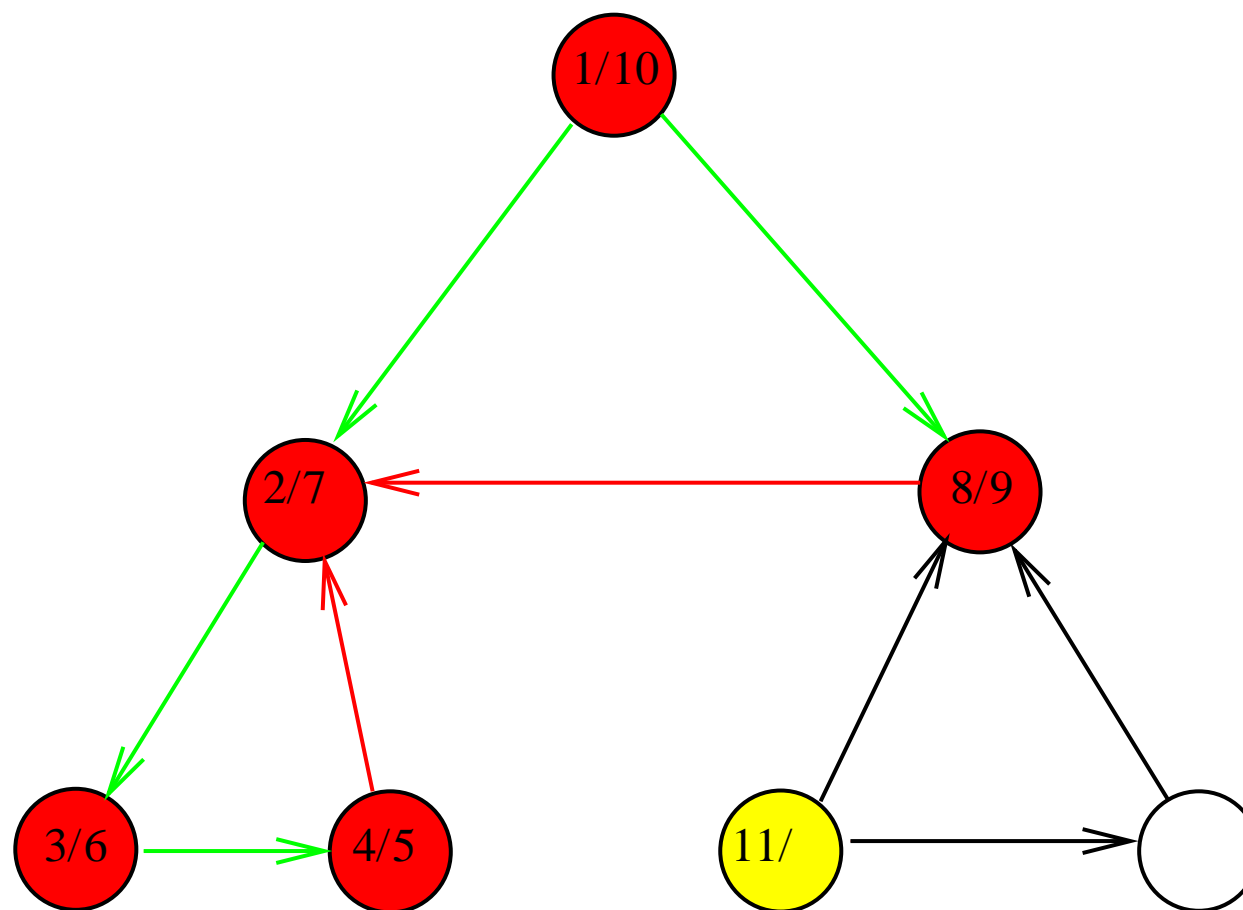


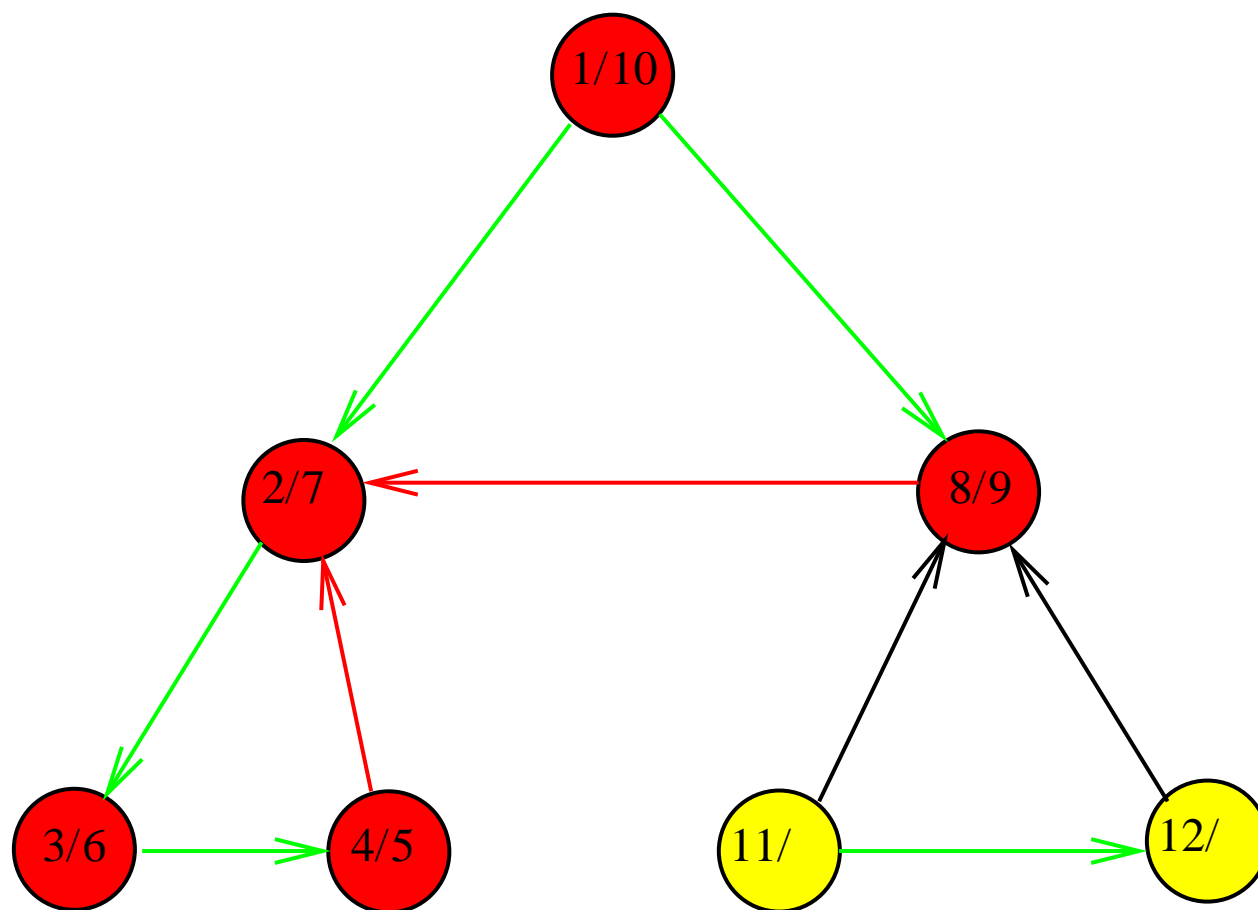


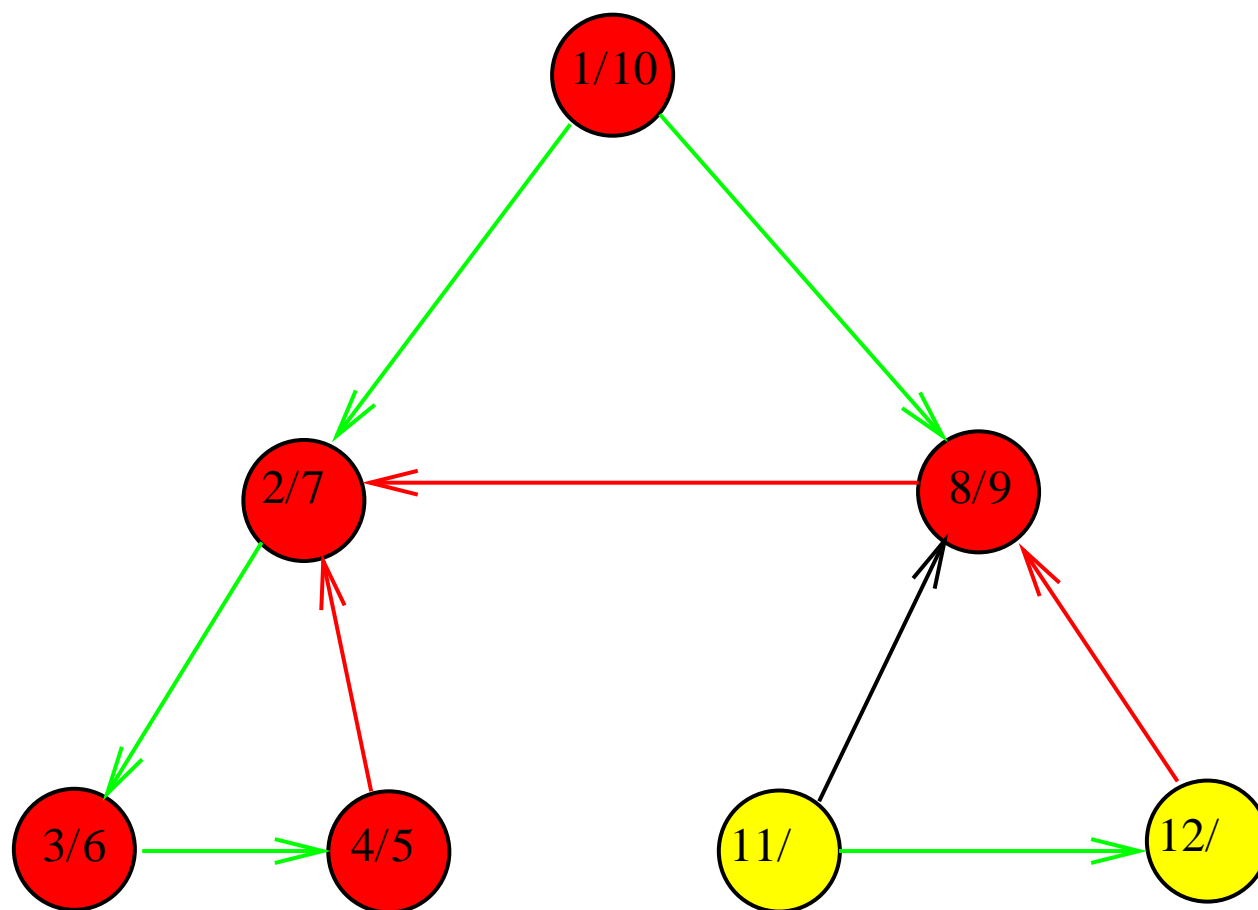


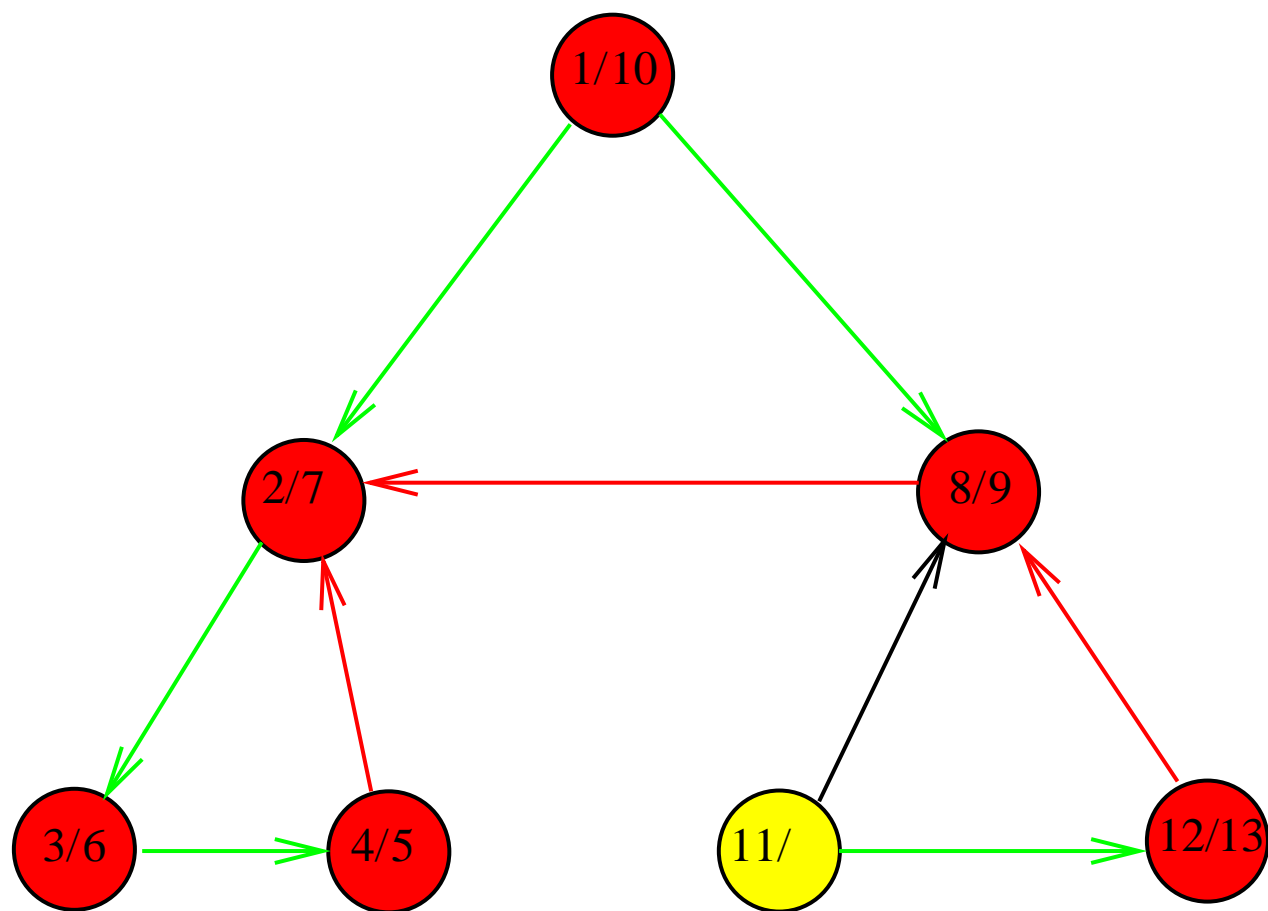


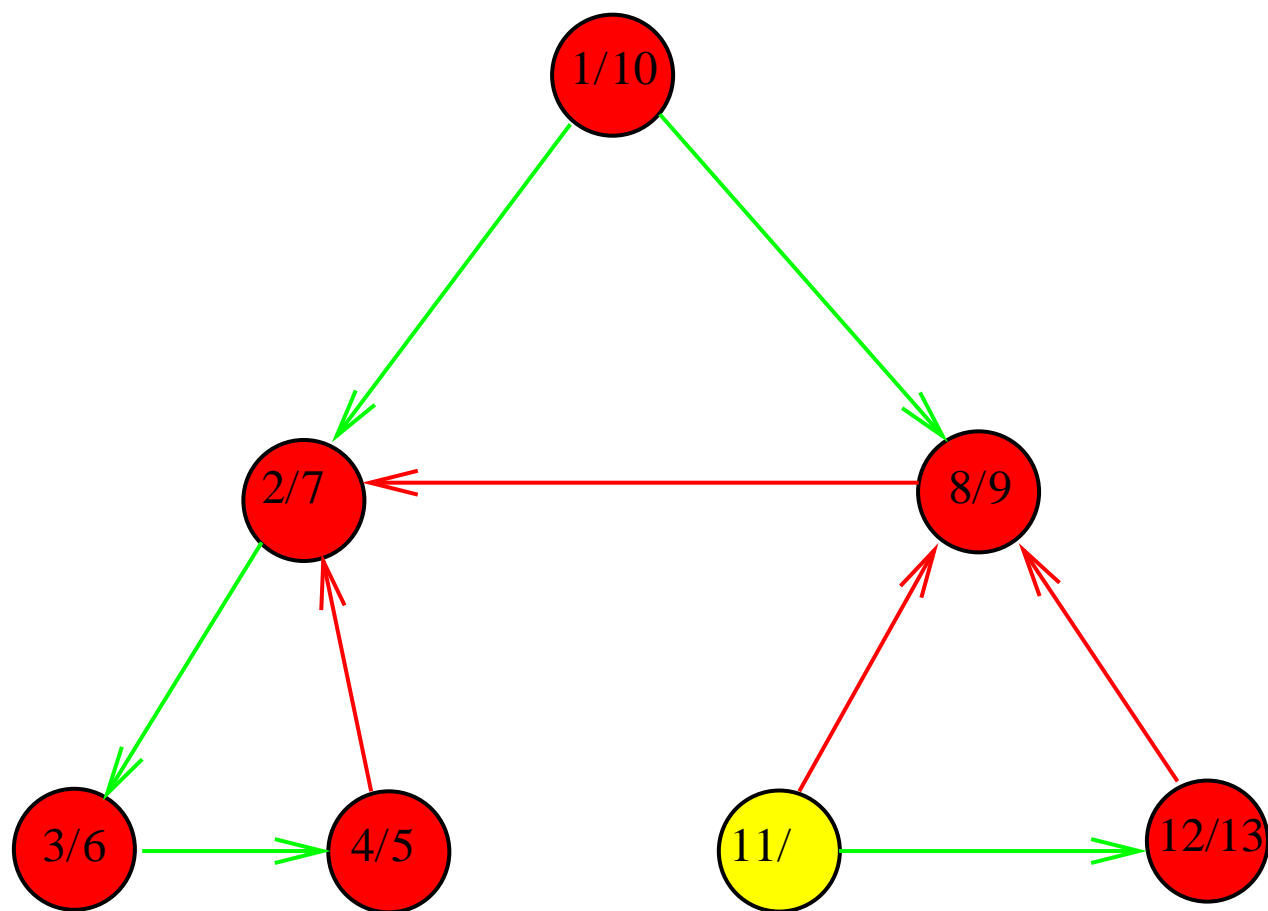


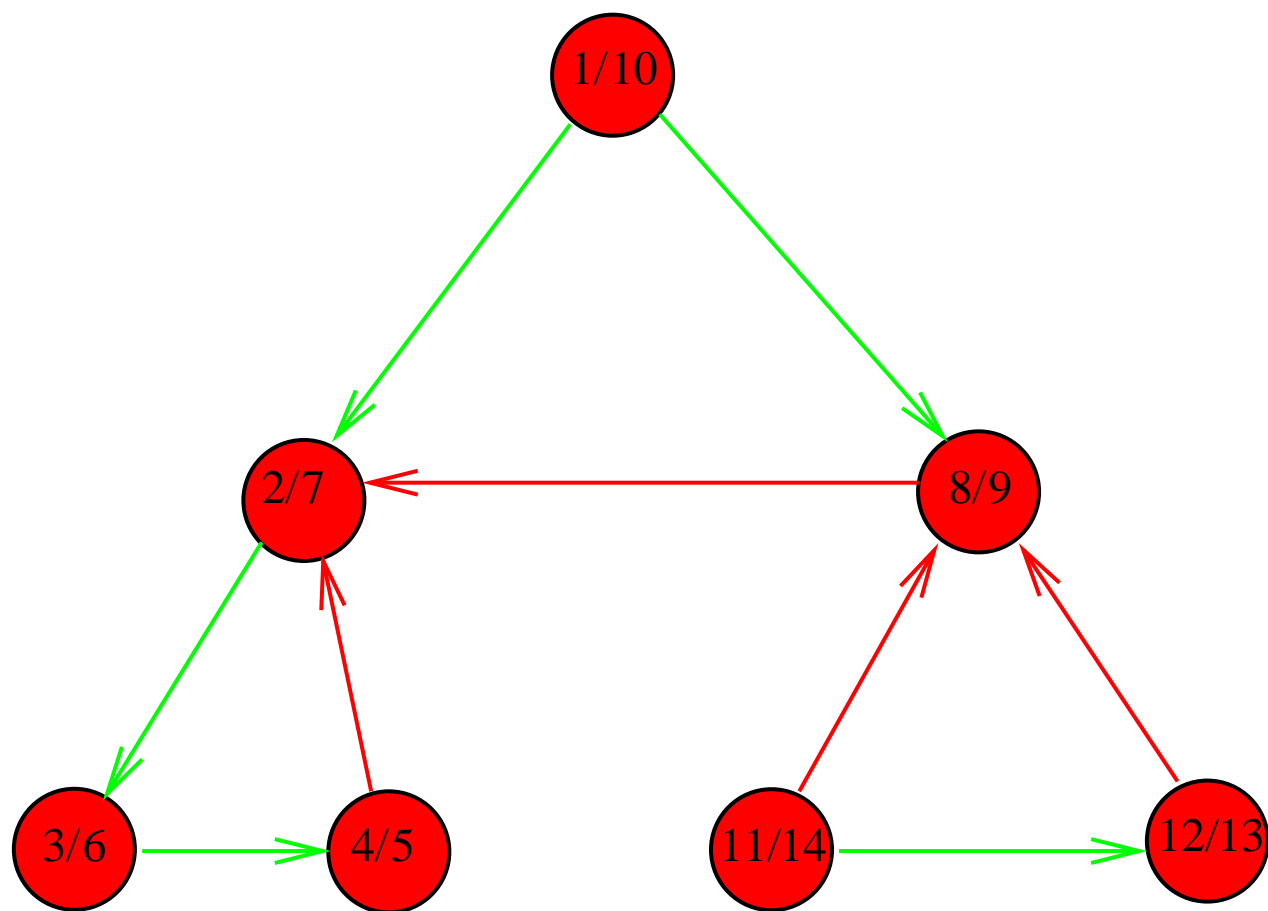












Algorytm DFS w wersji rekurencyjnej

Algorithm 0.0.1: DFS(G, x)

global $Numer, Drzewo, Pozostałe$

$v \leftarrow x$

$Numer[v] \leftarrow Ponumerowano \leftarrow 1$

for each $w \in \Gamma(v)$

do if $Numer[w] = 0$

then $\begin{cases} Ponumerowano \leftarrow Ponumerowano + 1 \\ Numer[w] \leftarrow Ponumerowano \\ Drzewo \leftarrow Drzewo \cup \{vw\} \\ DFS(G, w) \end{cases}$

else if $Numer(w) < Numer(v)$

then $Pozostałe \leftarrow Pozostałe \cup \{vw\}$

return ($Numer, Drzewo, Pozostałe$)

Przykładem zastosowania procedury *DFS* jest algorytm NUMEROWANIEWSZYSTKICHWIERZCHOŁKÓW, który wykorzystuje procedurę DFS do ponumerowania wszystkich wierzchołków grafu G .

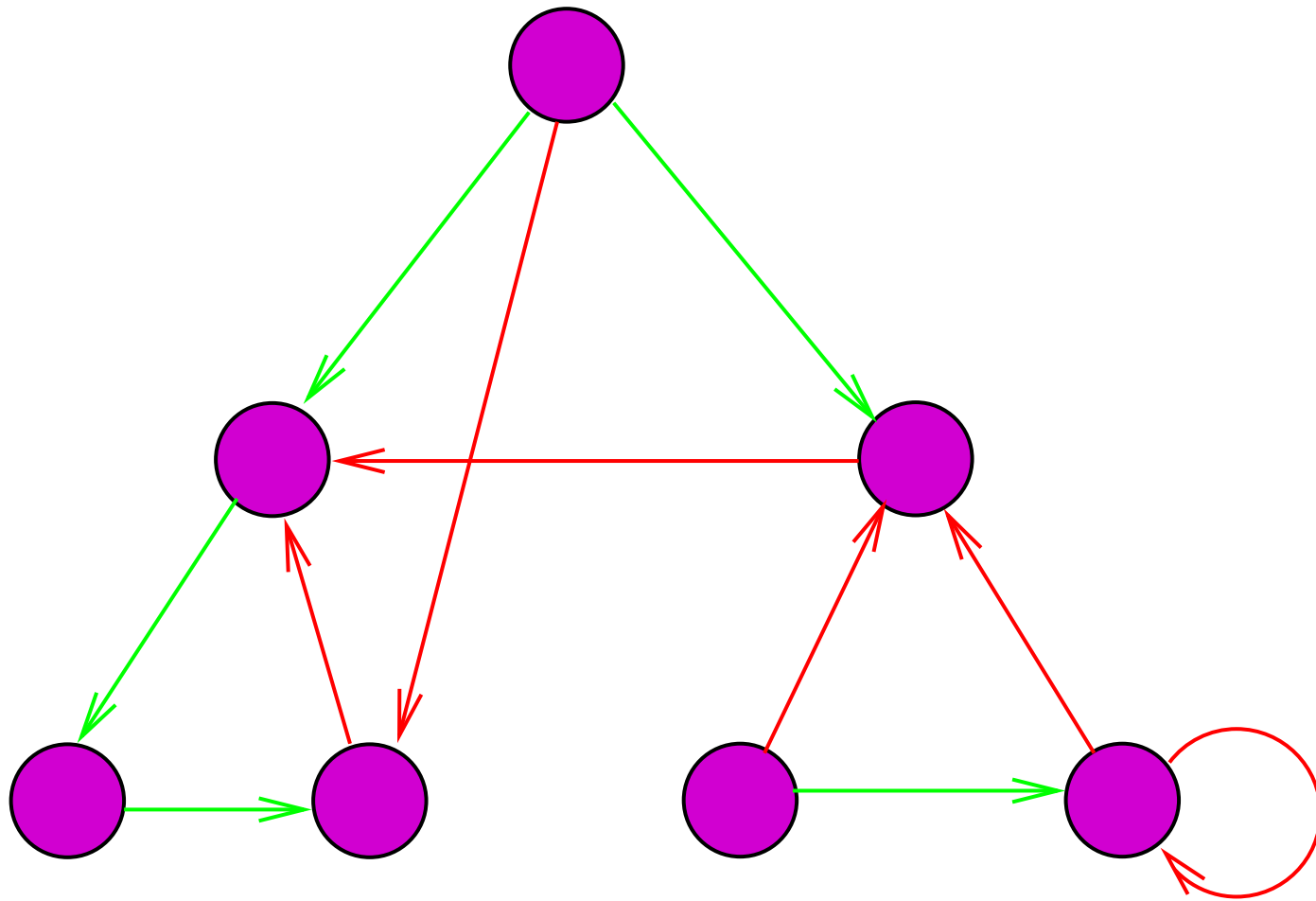
Algorithm 0.0.1: NUMEROWANIEWSZYSTKICHWIERZCHOŁKÓW(G)

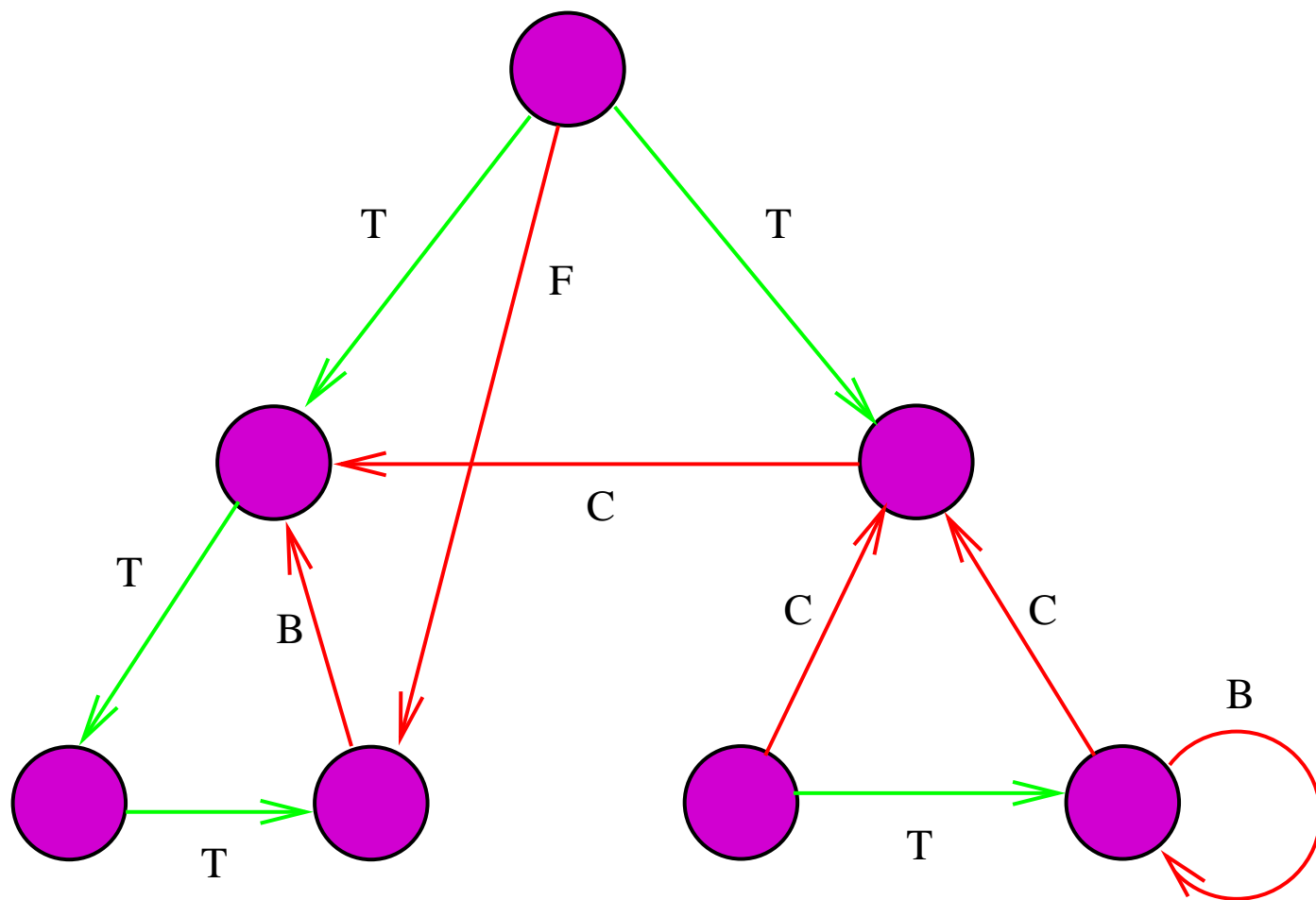
```
for each  $v \in V$   
  do  $Numer[v] \leftarrow 0$   
   $Drzewo \leftarrow Pozostałe \leftarrow \emptyset$   
for each  $v \in V$   
  do if  $Numer[v] = 0$  then DFS( $G, v$ )
```

Uwagi końcowe:

- złożoność DFS (i wielu algorytmów wykorzystujących DFS) jest rzędu $O(|V| + |E|)$
- dodatkowe informacje na temat poprawności obu algorytmów przeszukiwania grafów oraz ich zastosowań są zawarte Rozdziale 23 książki Cormena, Leisersona i Rivesta (patrz literatura pomocnicza wykładu)

Klasyfikacja krawędzi grafu przy przeszukiwaniu w głąb

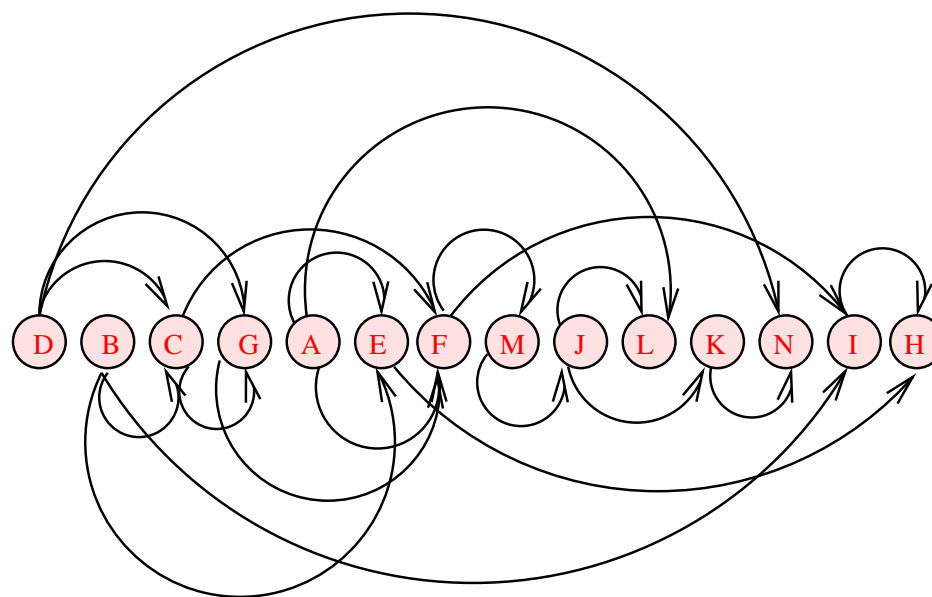
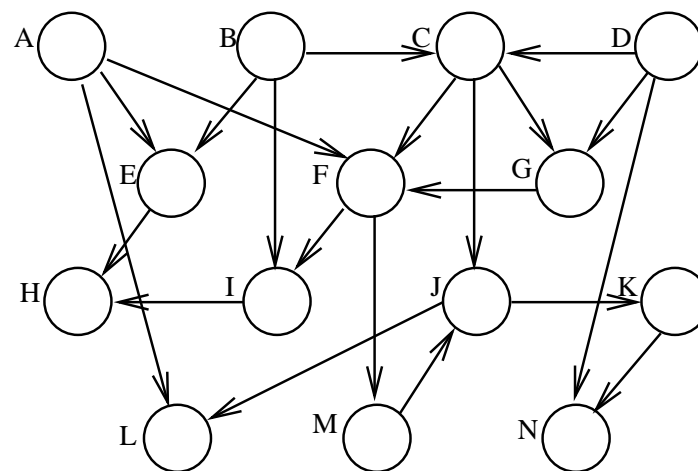




- **krawędzie drzewowe** (T) - krawędzie należące do drzew lasu przeszukiwań w głąb
- **krawędzie powrotne** (B) - krawędzie (u, v) łączące wierzchołek u z jego przodkiem v w drzewie
- **krawędzie w przód** (F) - krawędzie (u, v) łączące wierzchołek u z jego potomkiem v w drzewie
- **krawędzie poprzeczne** (C) - pozostałe krawędzie

Przykład . Sortowanie topologiczne

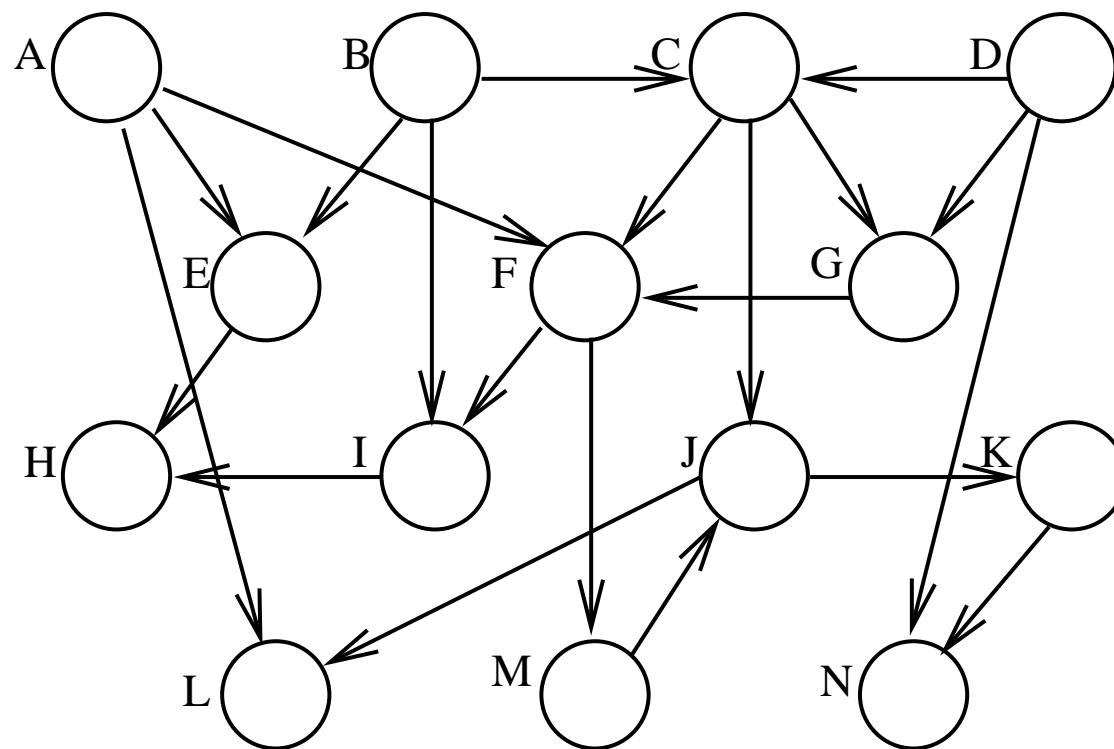
- $G = (V, E)$ - acykliczny graf skierowany
- (dag - directed acyclic graph)
- sortowanie topologiczne dagu polega na takim liniowym uporządkowaniu jego wierzchołków, że jeżeli dag zawiera łuk (u, v) to wierzchołek u występuje w tym uporządkowaniu przed wierzchołkiem v
- równoważnie, jeżeli $G = (V, E)$ jest dagiem to można umieścić jego wierzchołki na prostej, w taki sposób, że wszystkie łuki są skierowane od lewej do prawej

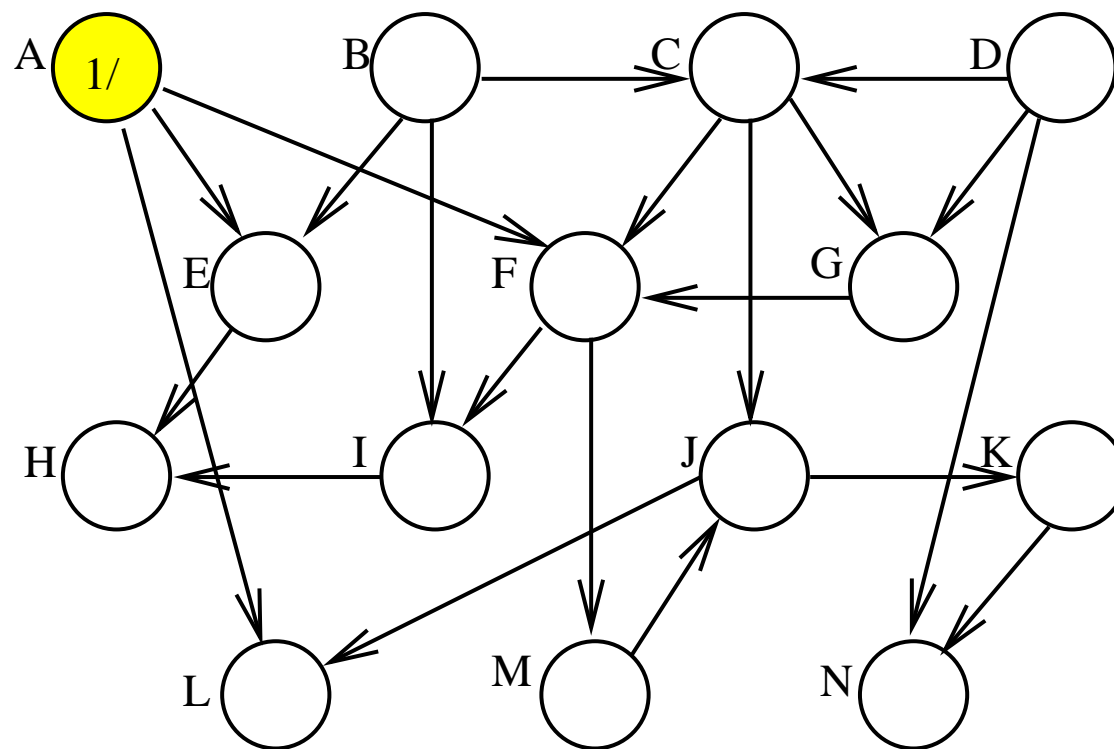


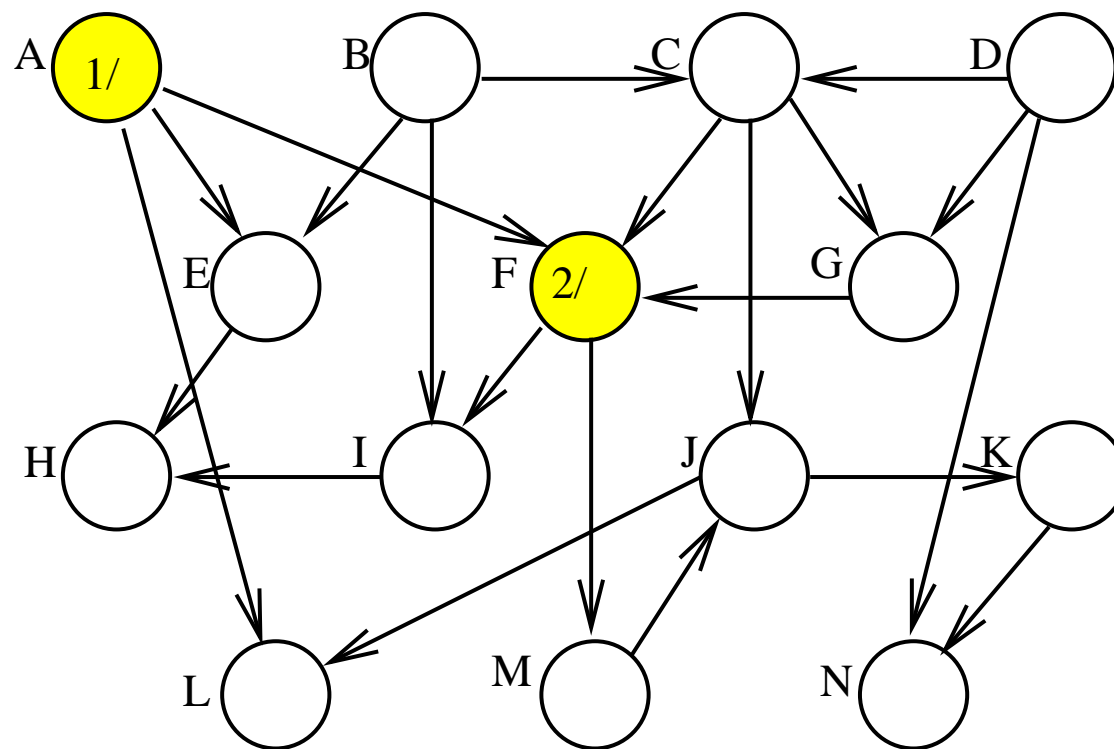
Lemat . Graf skierowany G jest grafem acyklicznym wtedy i tylko wtedy, gdy przy przeszukiwaniu w głąb grafu G nie pojawiają się krawędzie *powrotne*.

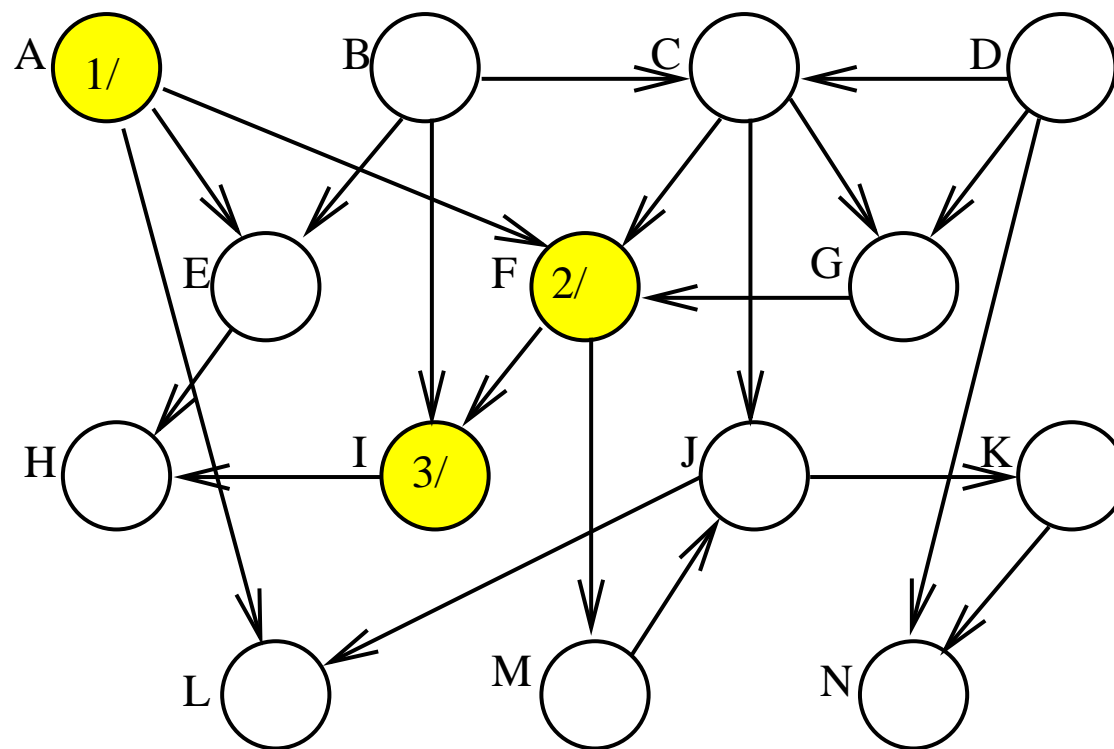
SORTOWANIE-TOPOLOGICZNE(G)

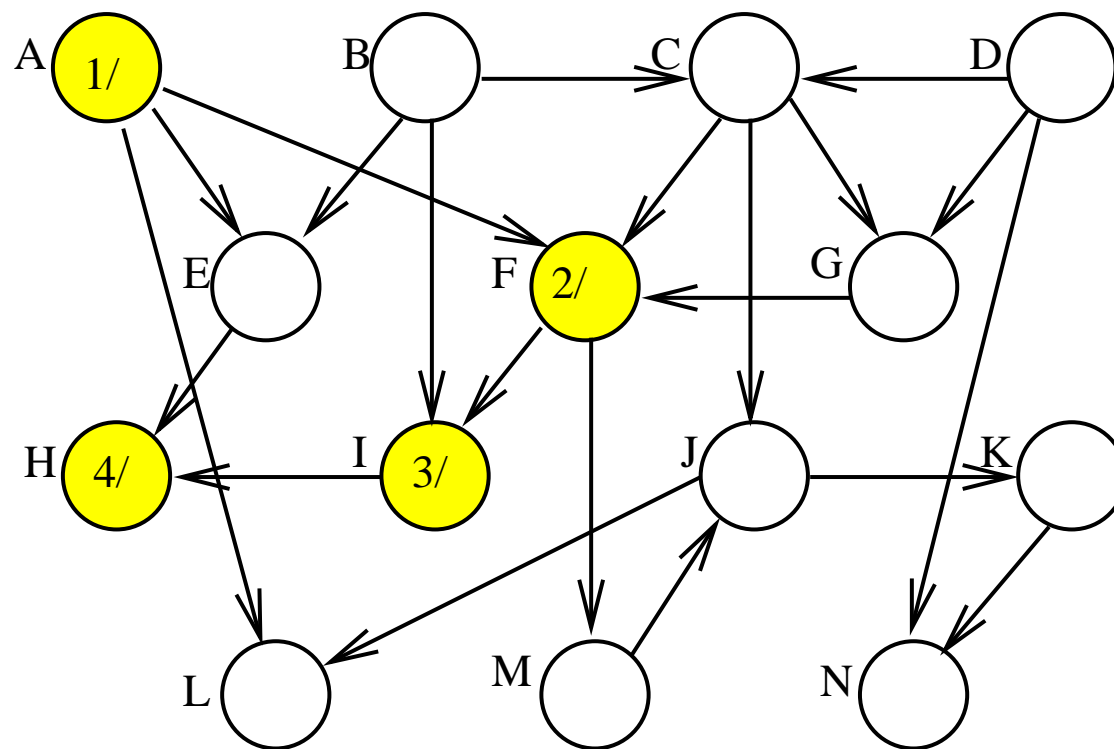
- wykonaj $DFS(G)$ w celu obliczenia czasów *przetworzenia* dla wszystkich wierzchołków
- wstaw wierzchołek v na początek listy, kiedy tylko zostanie on przetworzony

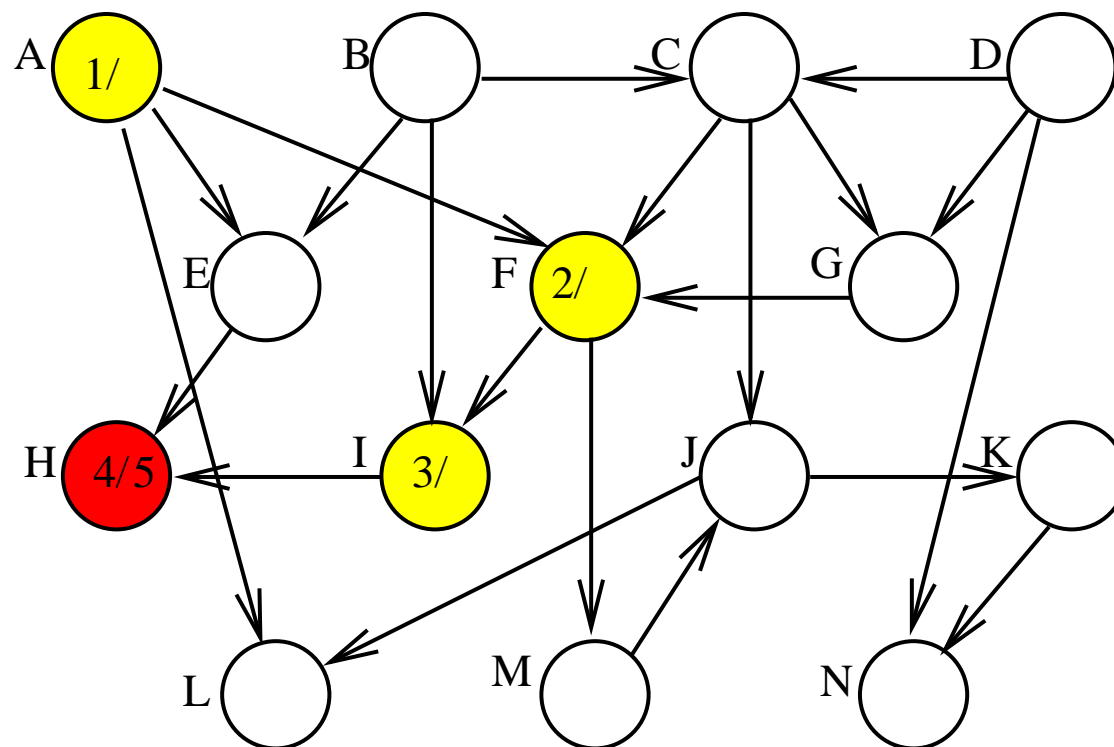




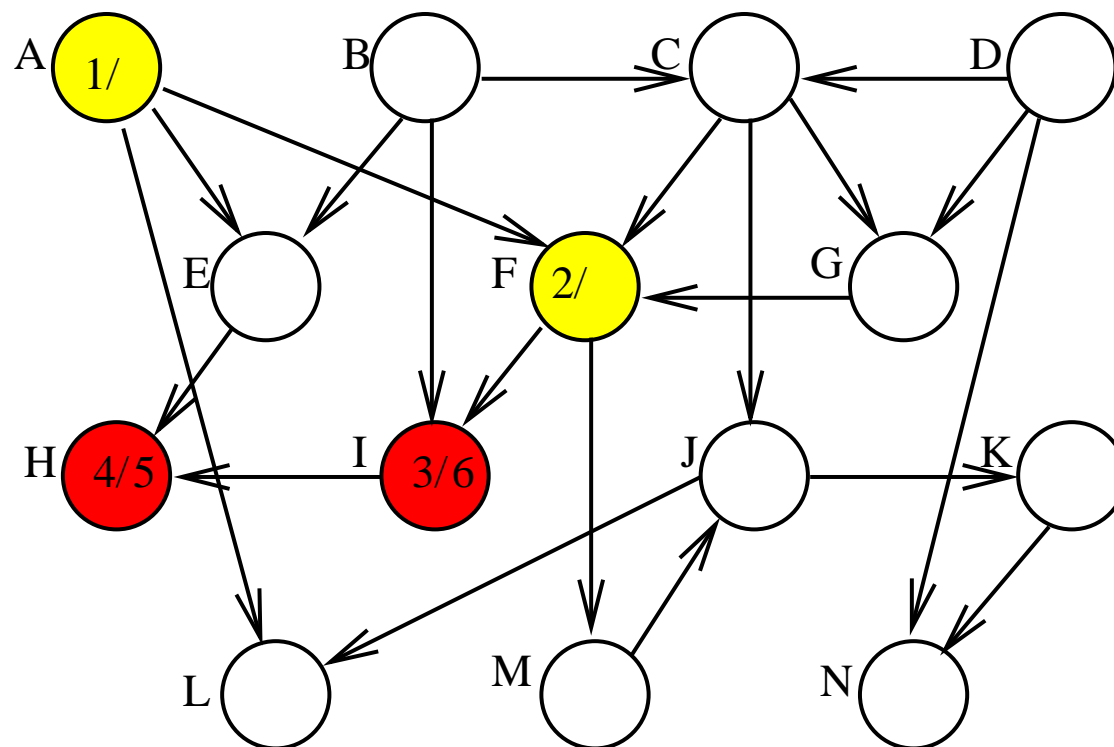




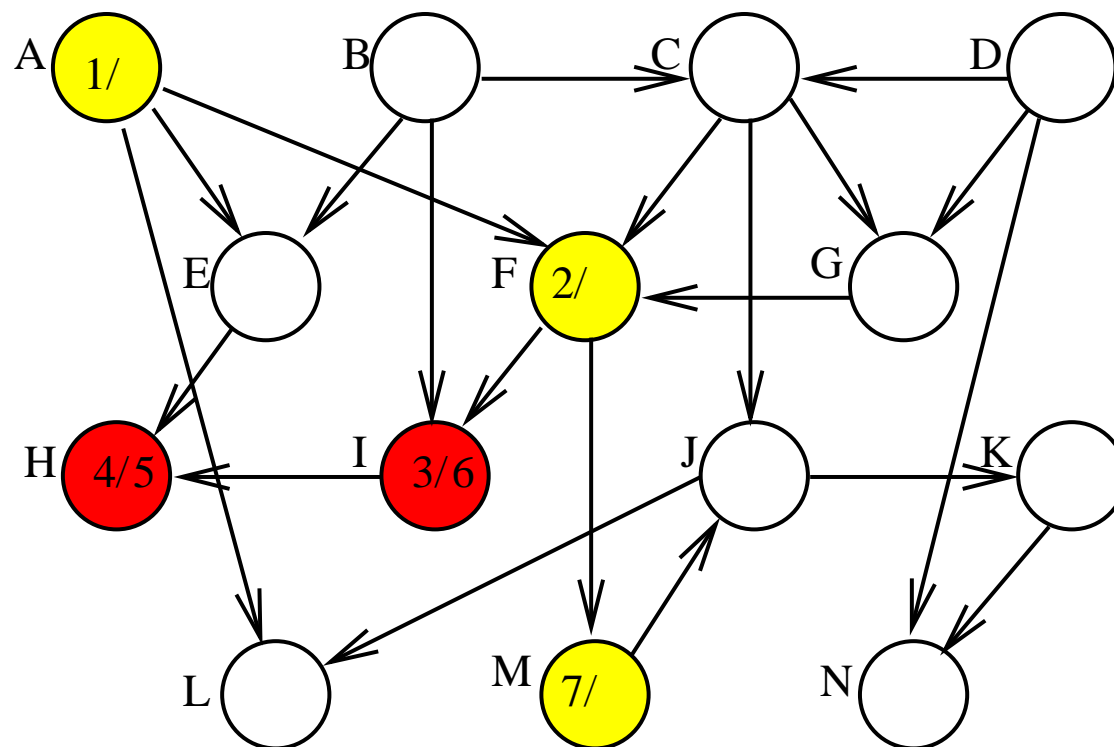




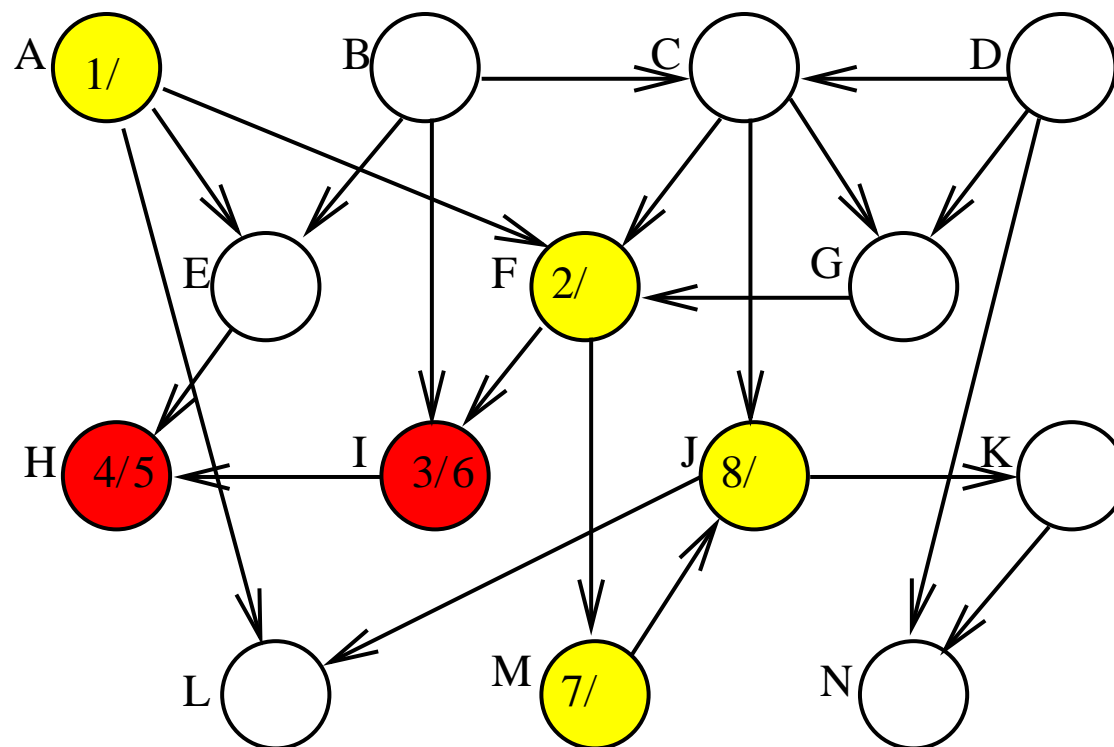
Kolejka: **H**



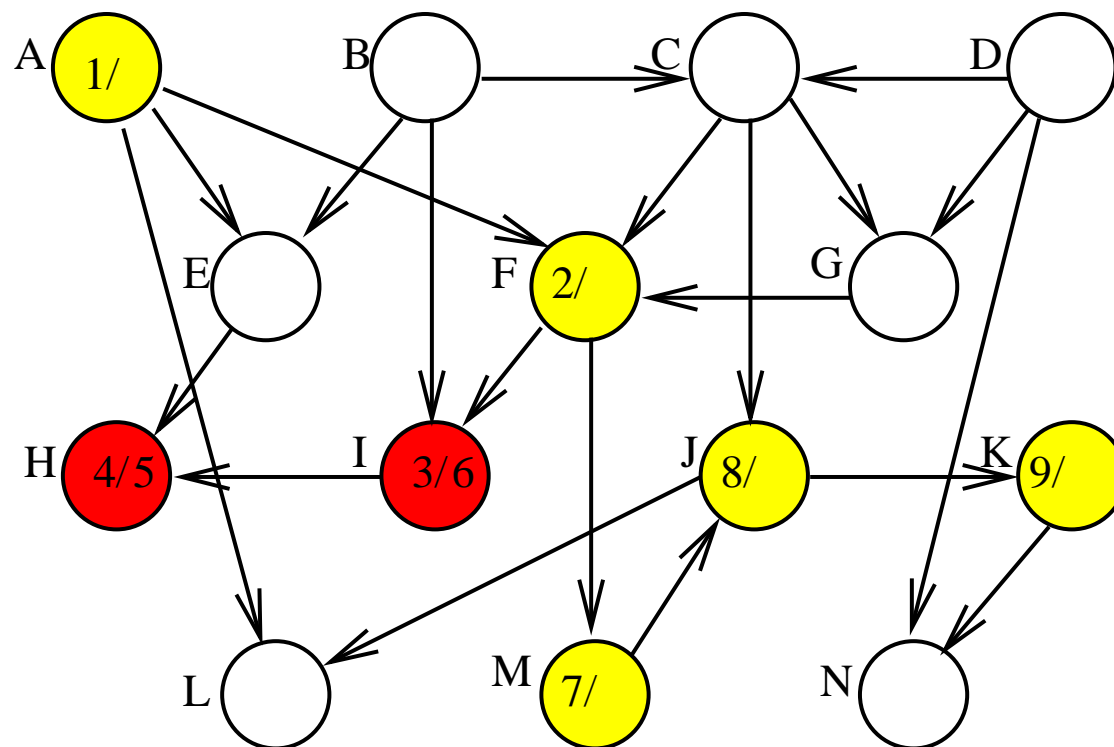
Kolejka: I, H



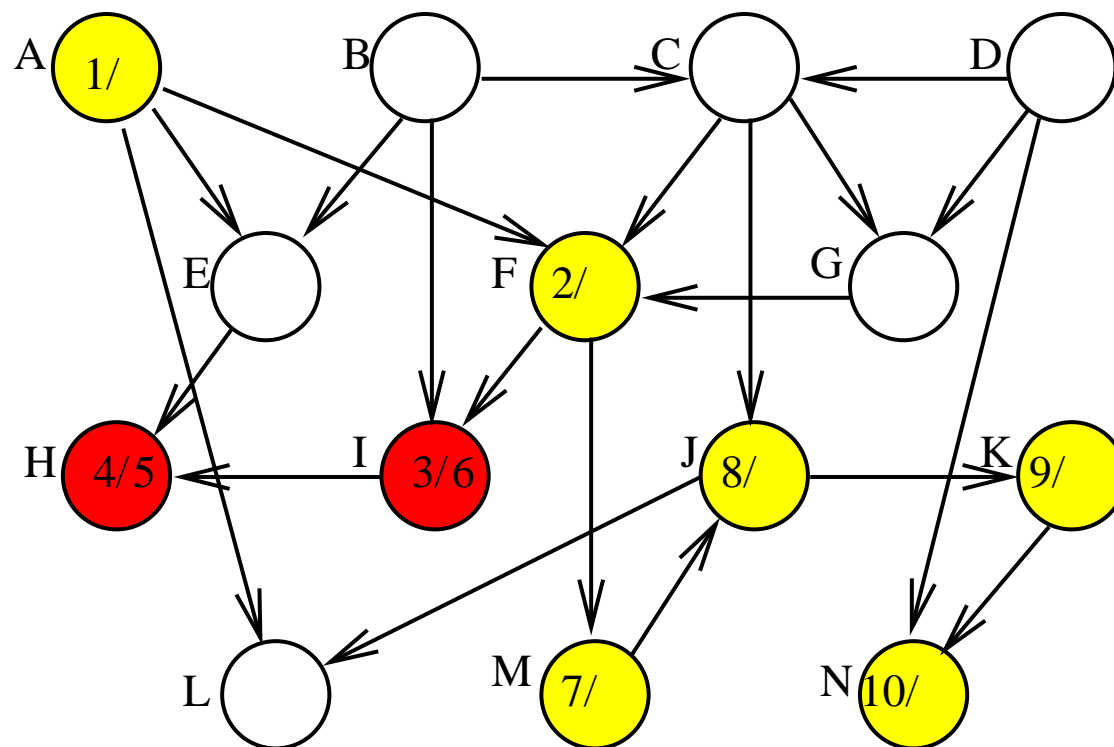
Kolejka: **I, H**



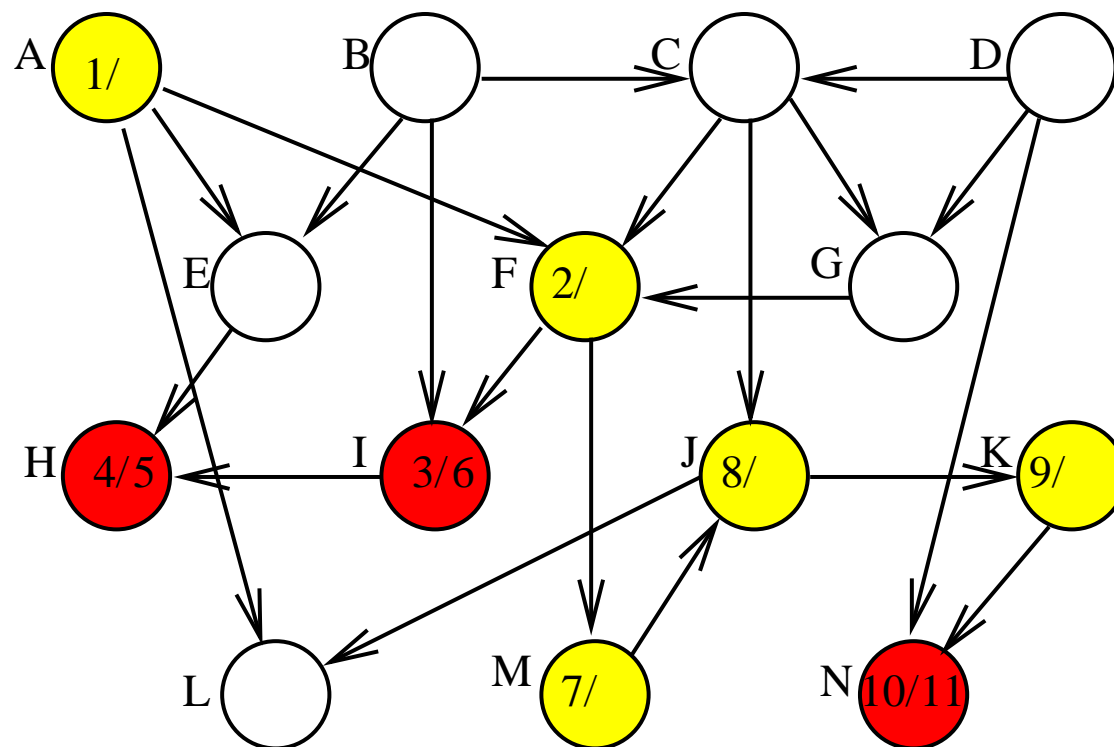
Kolejka: **I, H**



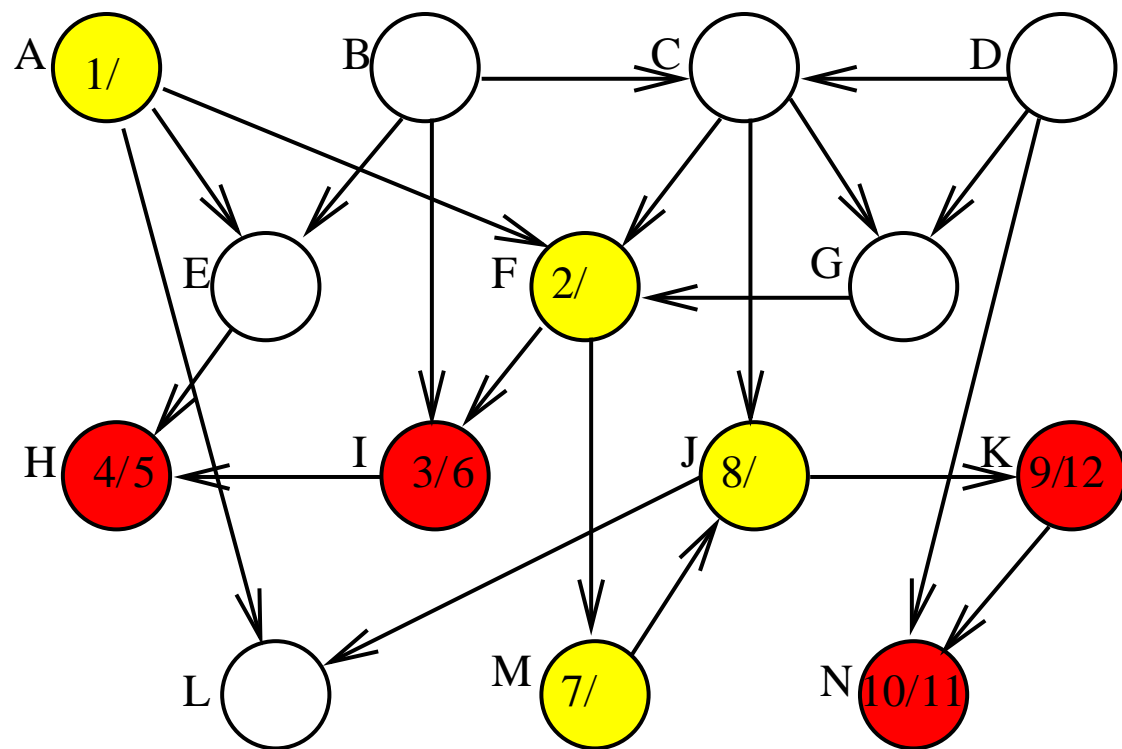
Kolejka: **I, H**



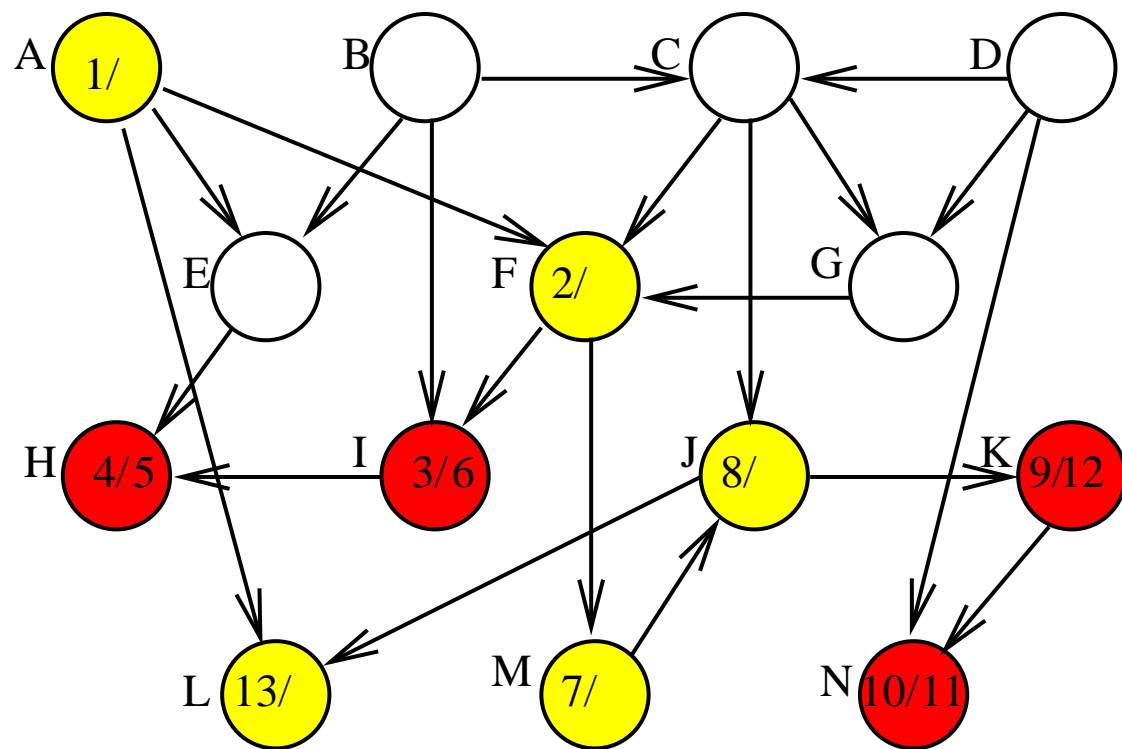
Kolejka: **I, H**



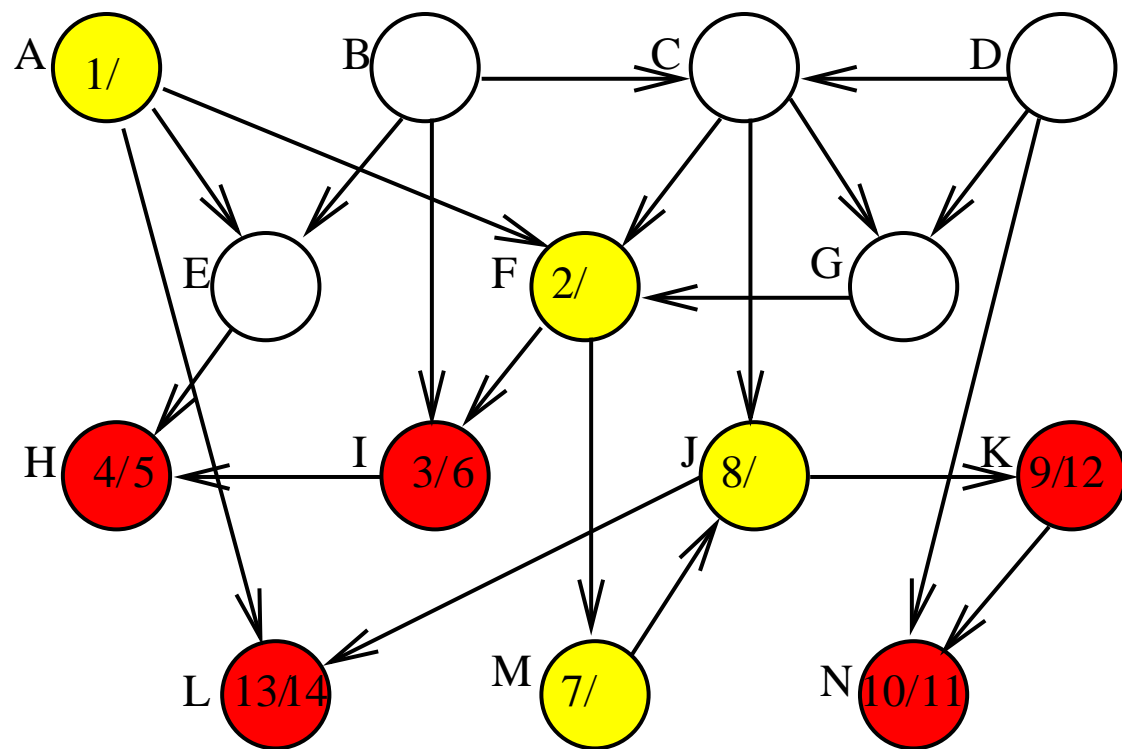
Kolejka: **N, I, H**



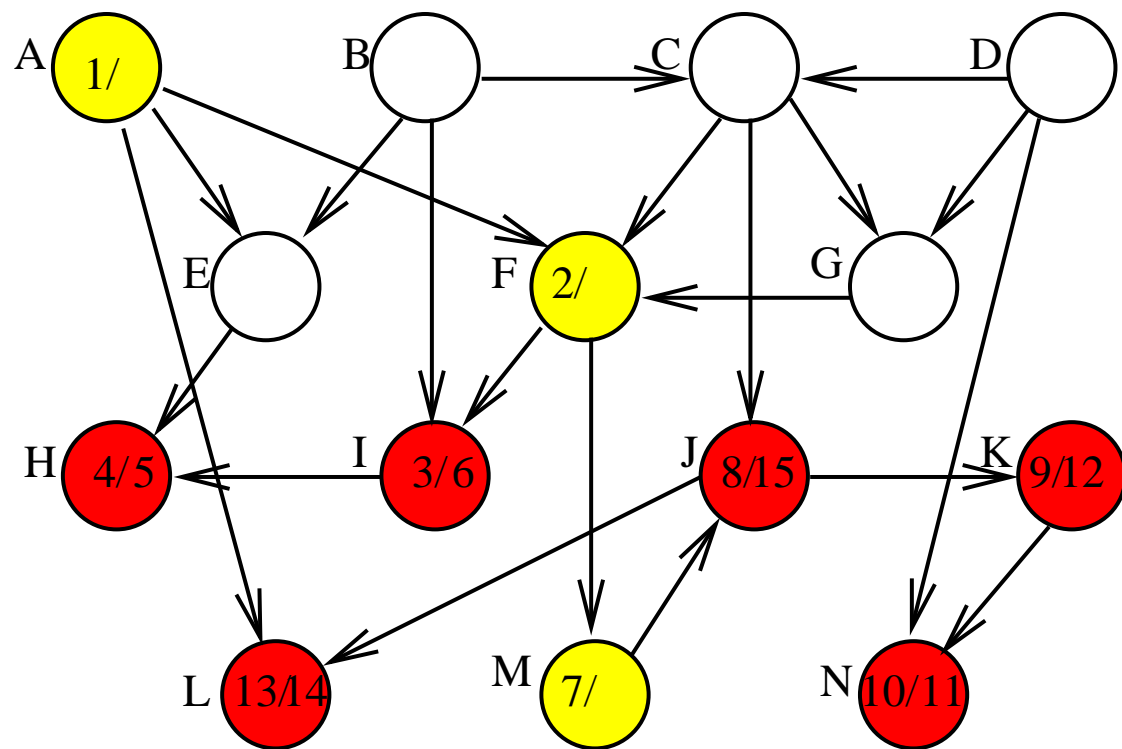
Kolejka: **K, N, I, H**



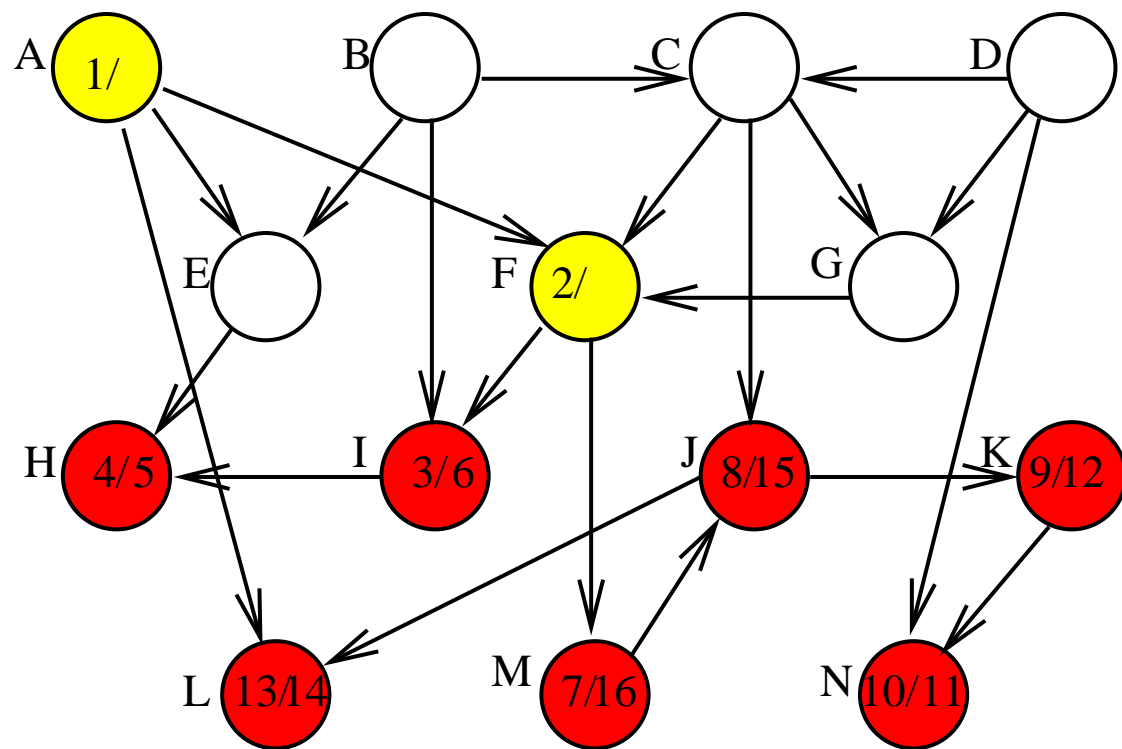
Kolejka: **K, N, I, H**



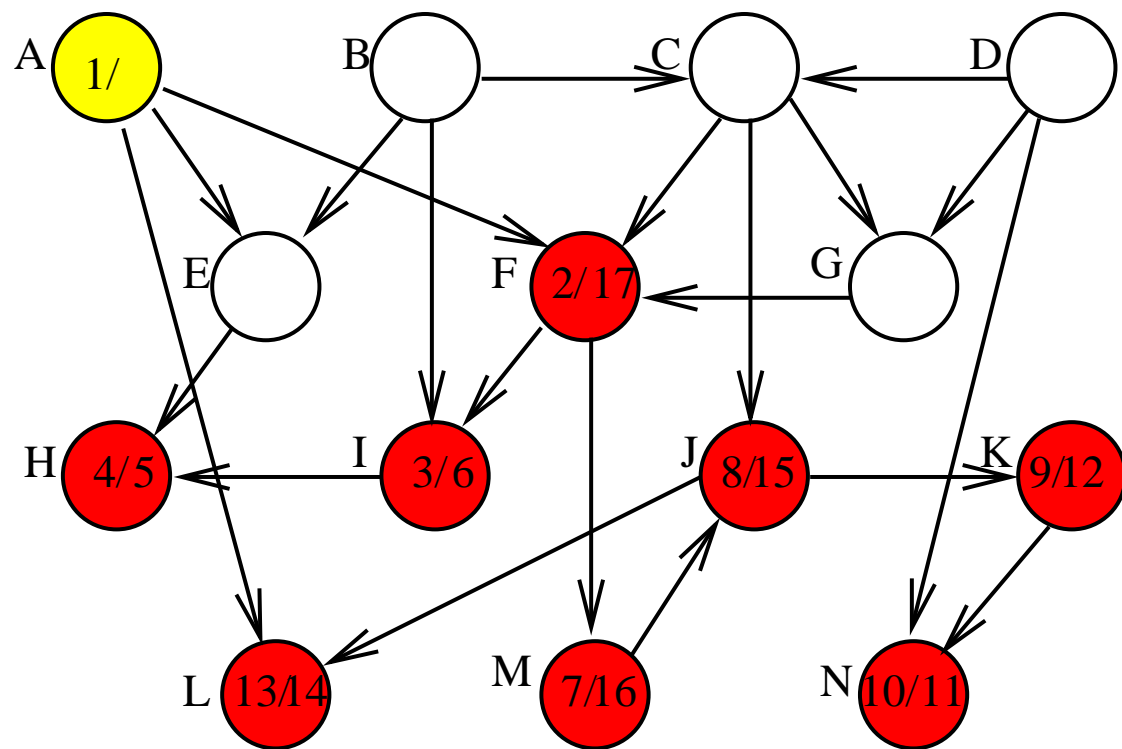
Kolejka: L, K, N, I, H



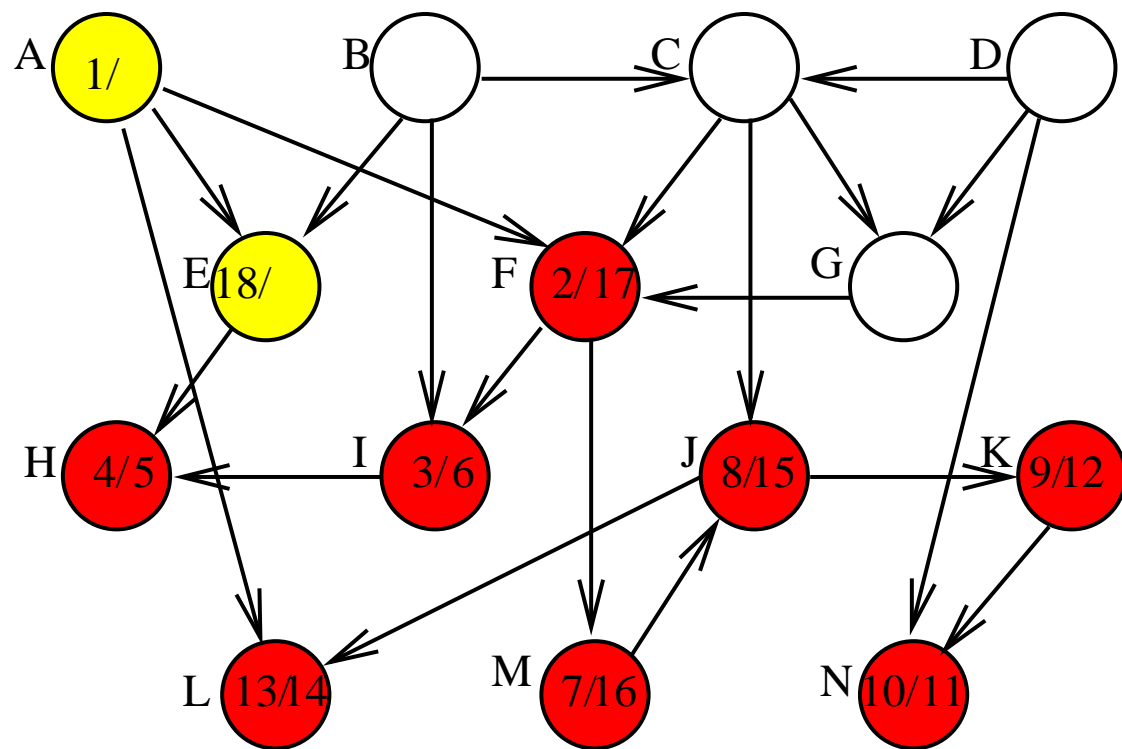
Kolejka: J, L, K, N, I, H



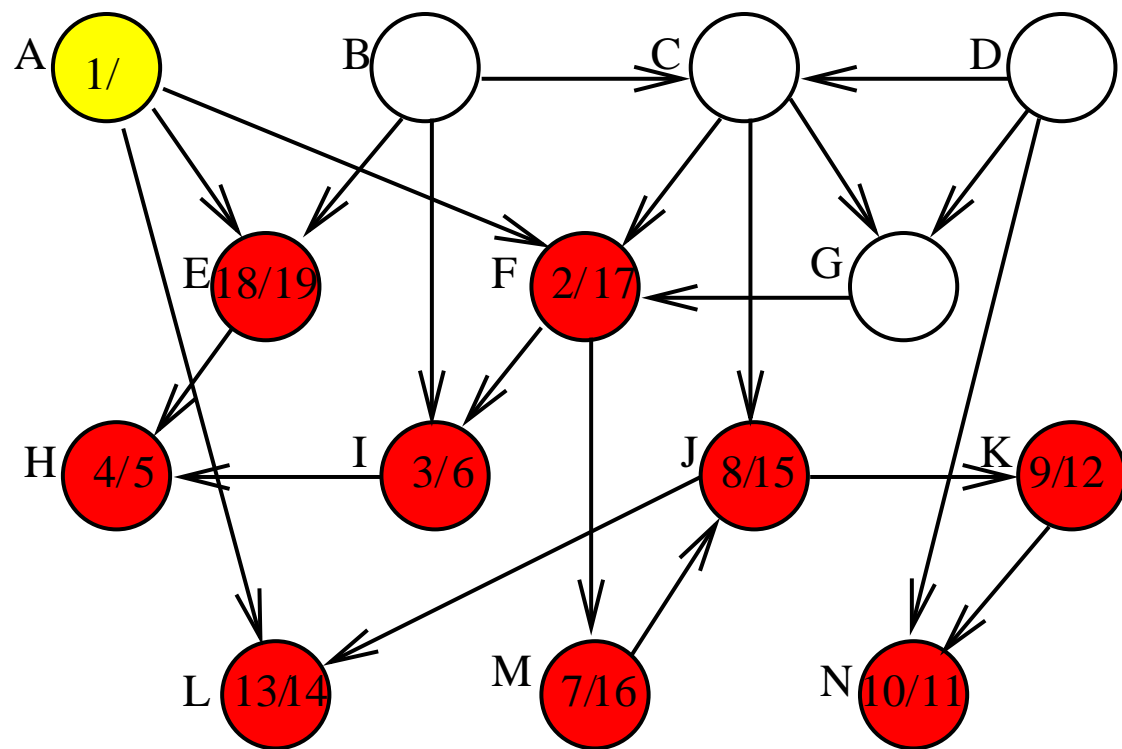
Kolejka: M, J, L, K, N, I, H



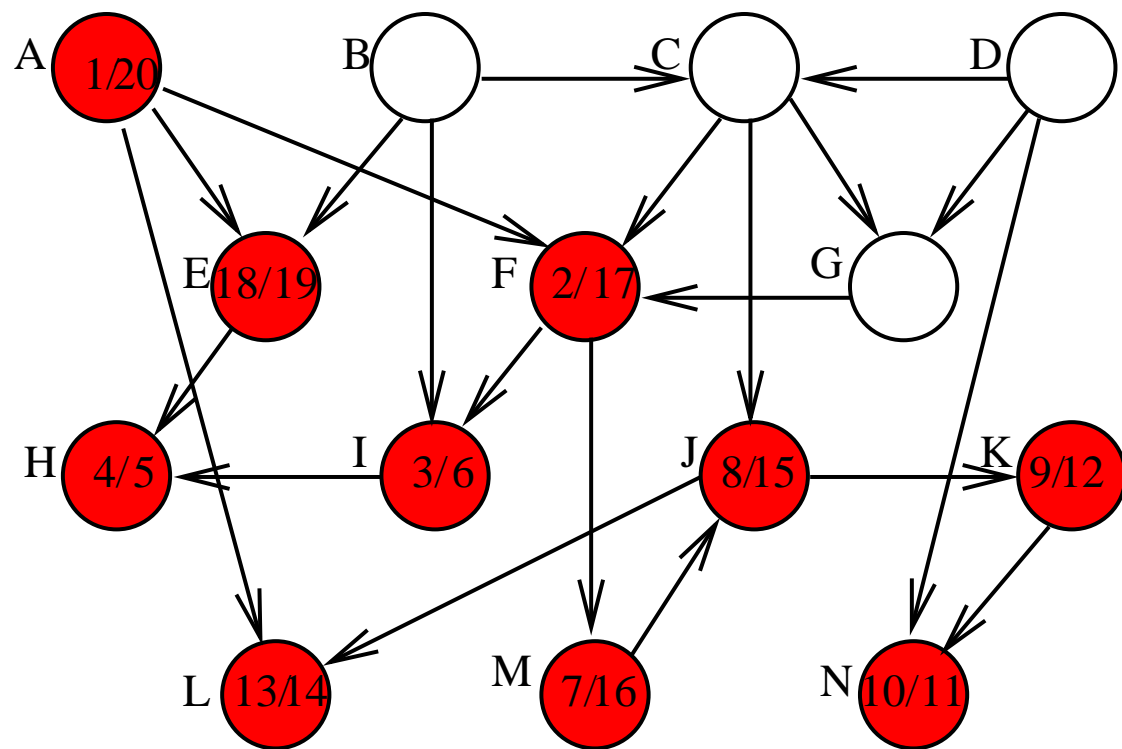
Kolejka: F, M, J, L, K, N, I, H



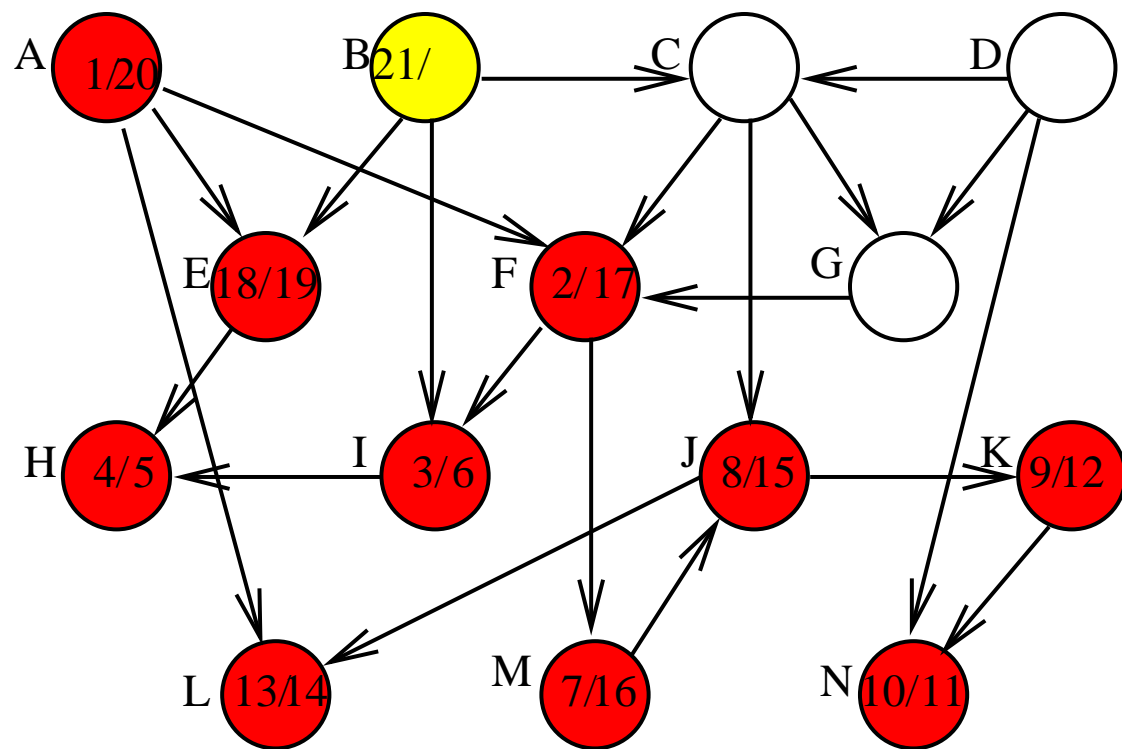
Kolejka: F, M, J, L, K, N, I, H



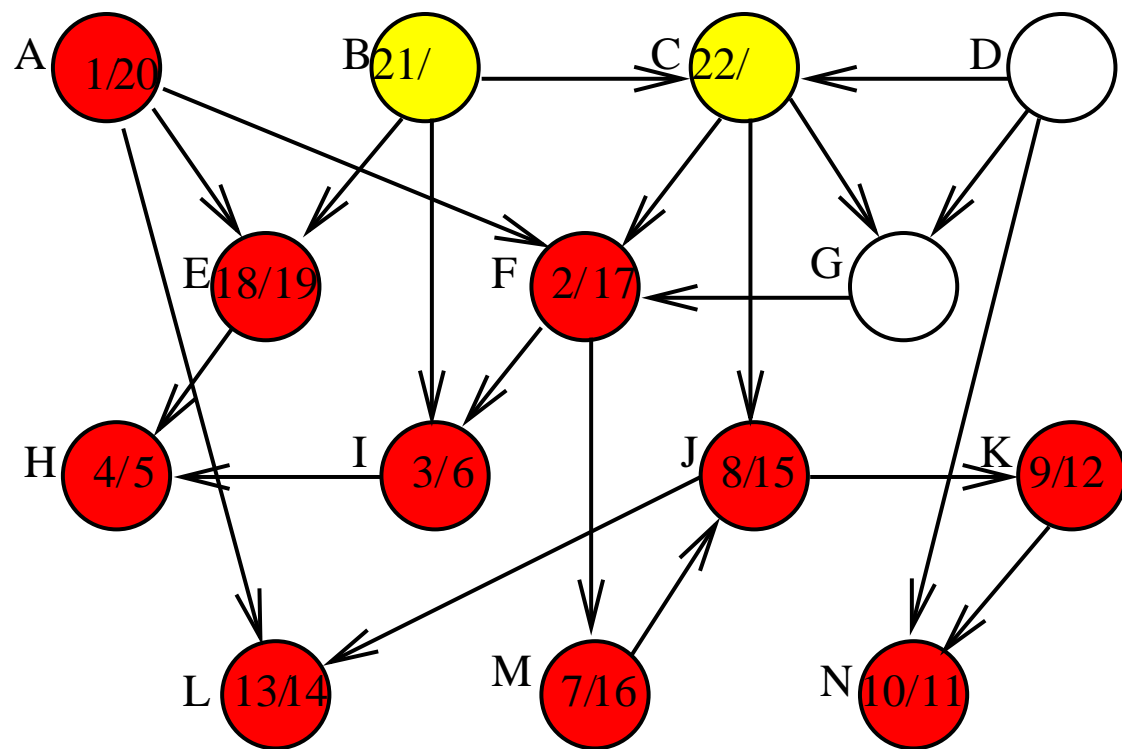
Kolejka: E, F, M, J, L, K, N, I, H



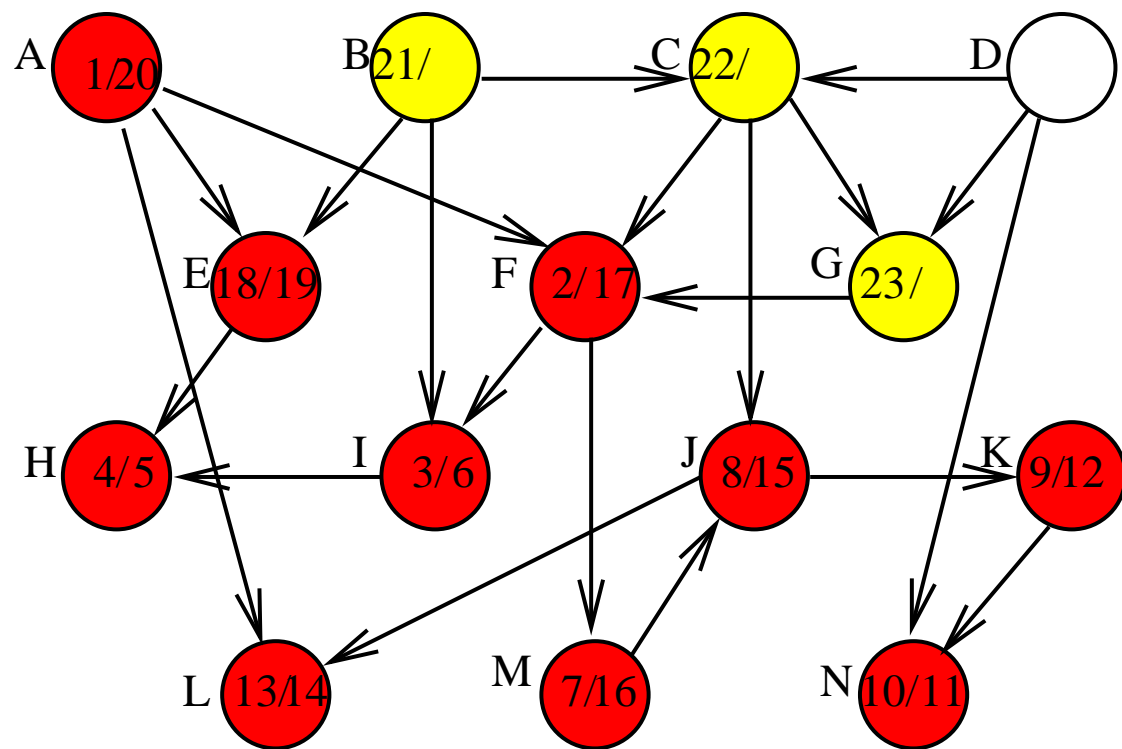
Kolejka: A, E, F, M, J, L, K, N, I, H



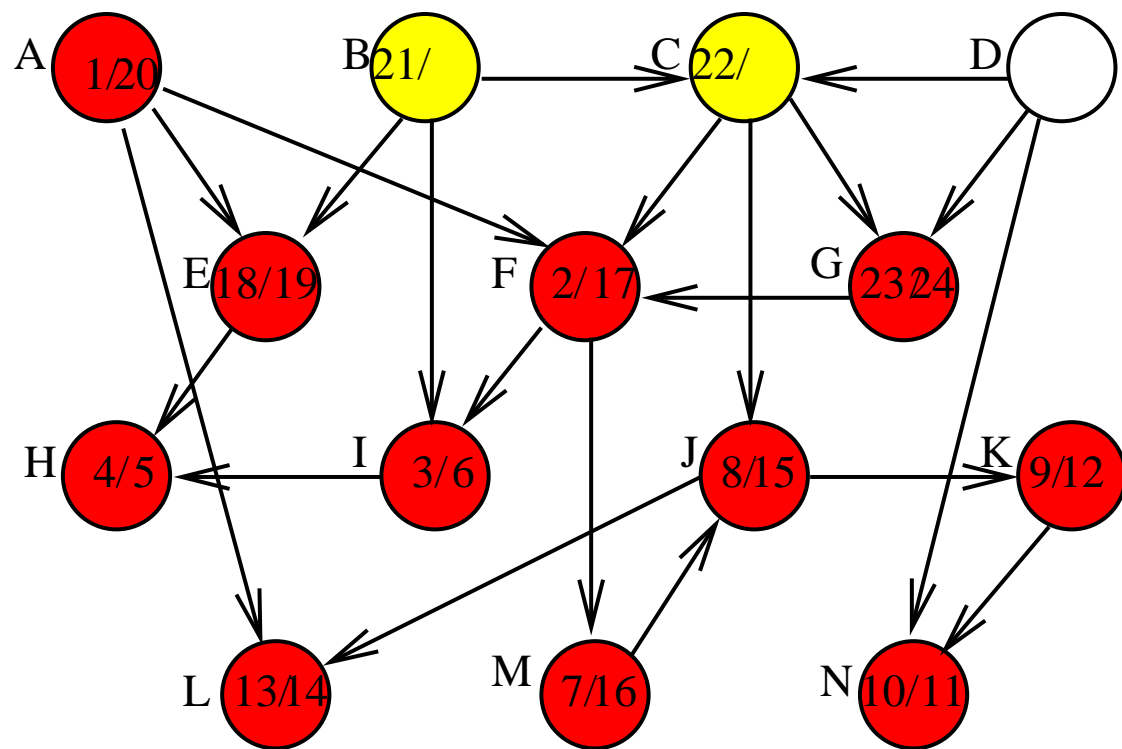
Kolejka: A, E, F, M, J, L, K, N, I, H



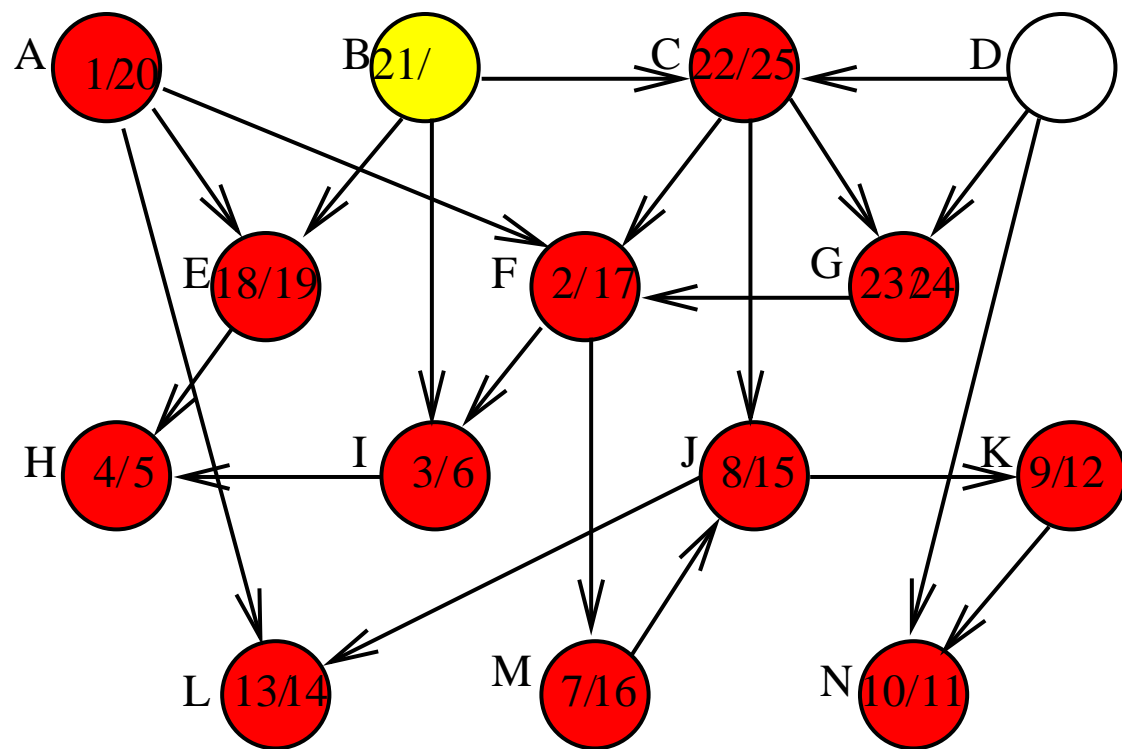
Kolejka: A, E, F, M, J, L, K, N, I, H



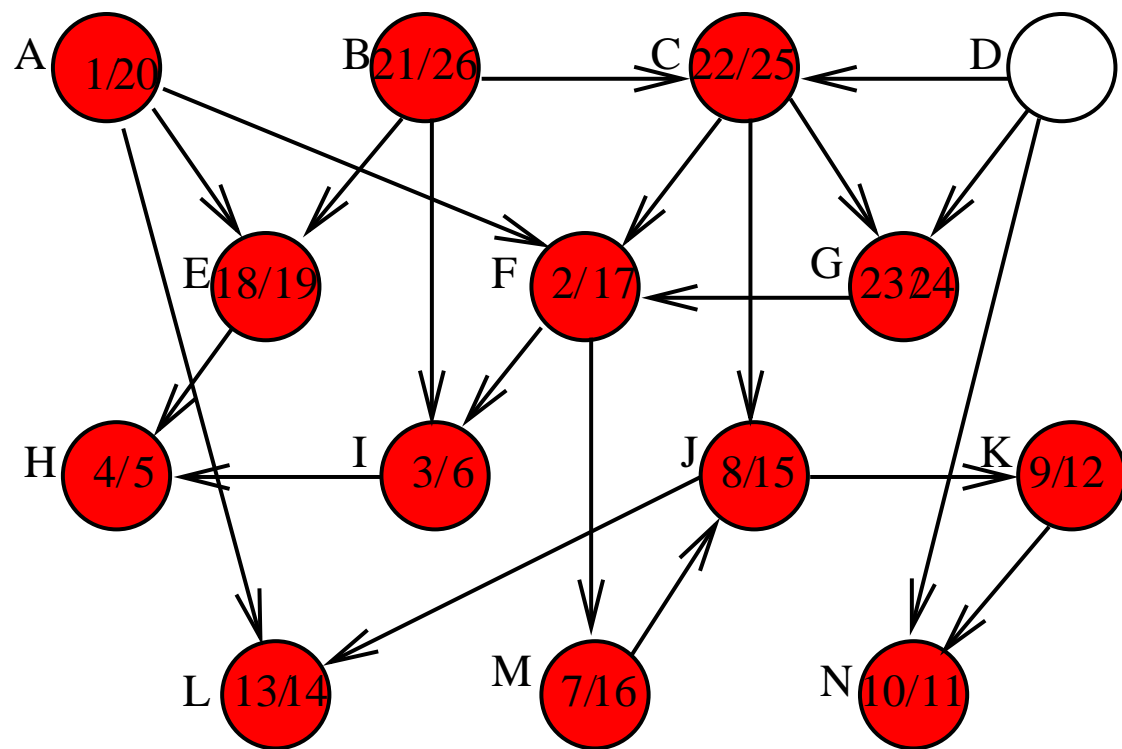
Kolejka: A, E, F, M, J, L, K, N, I, H



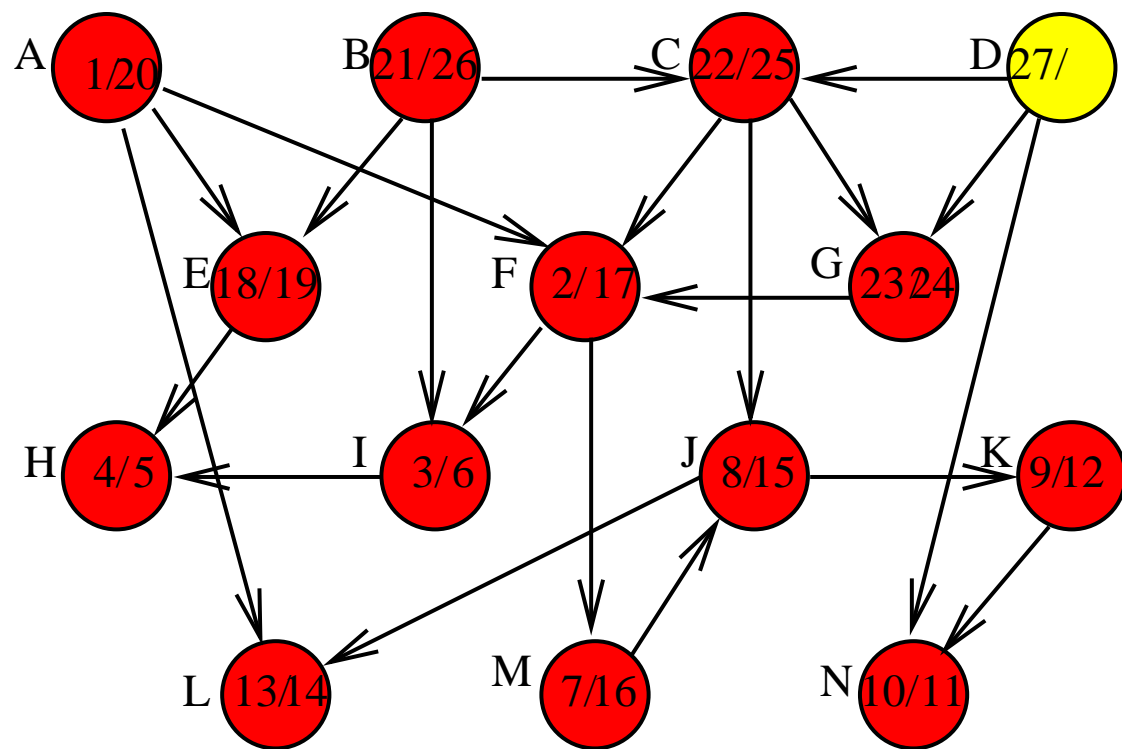
Kolejka: G, A, E, F, M, J, L, K, N, I, H



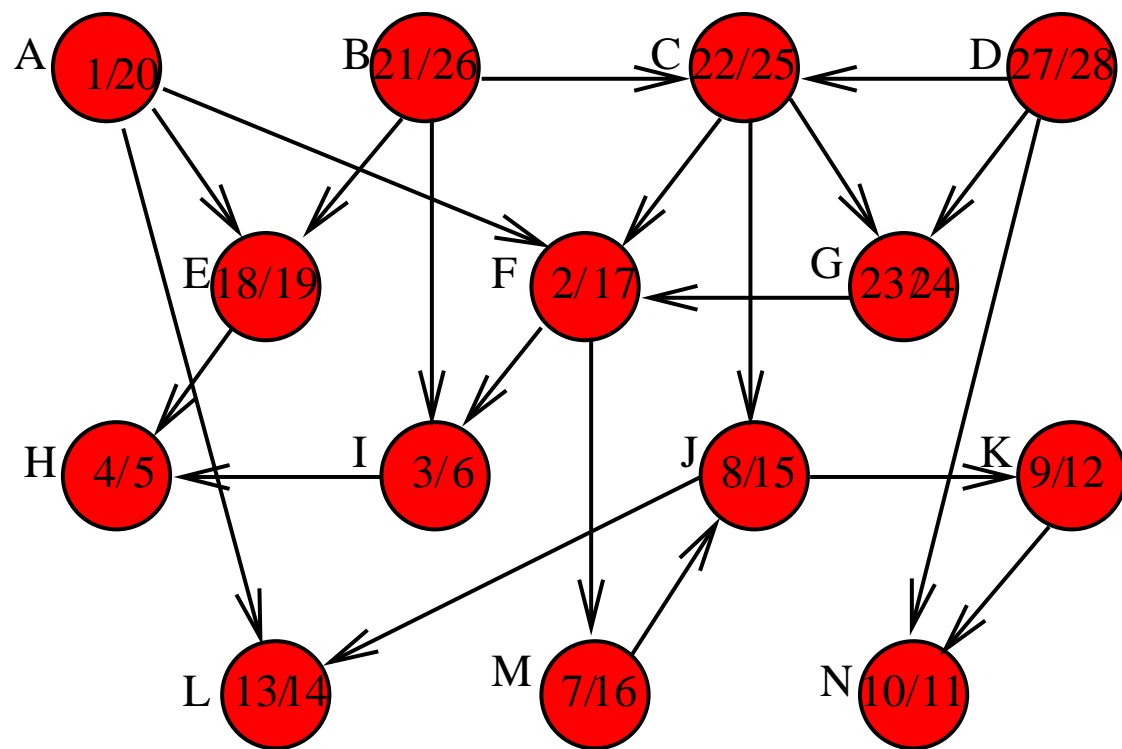
Kolejka: C, G, A, E, F, M, J, L, K, N, I, H



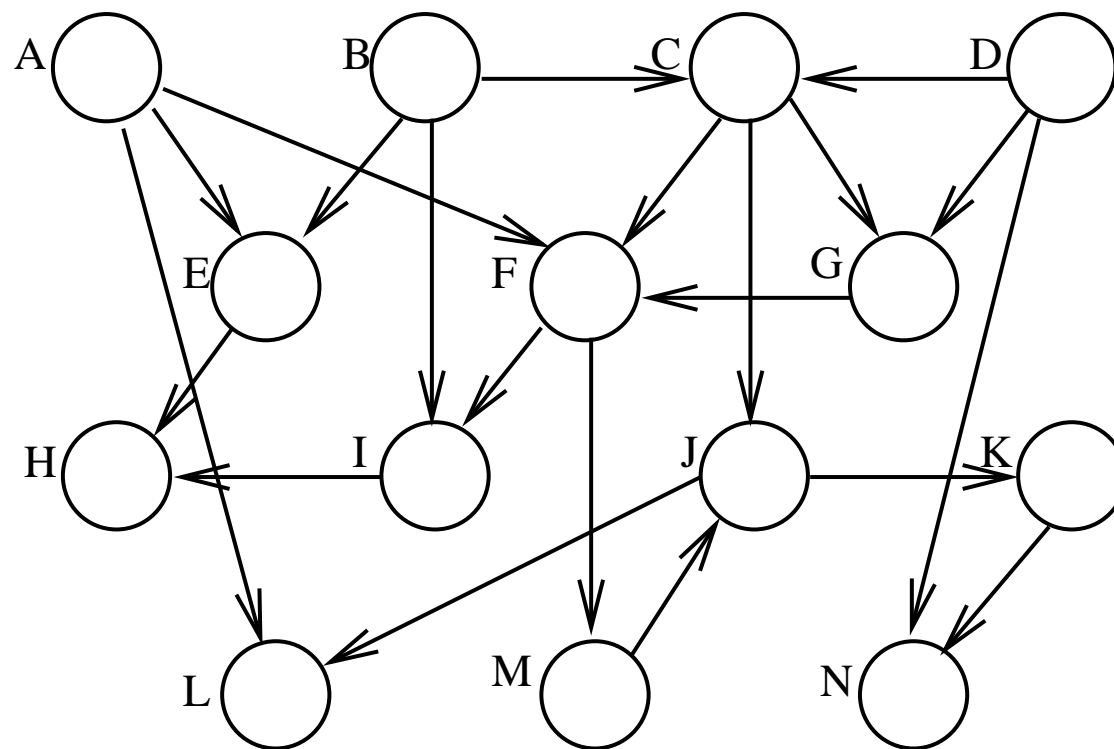
Kolejka: B, C, G, A, E, F, M, J, L, K, N, I, H

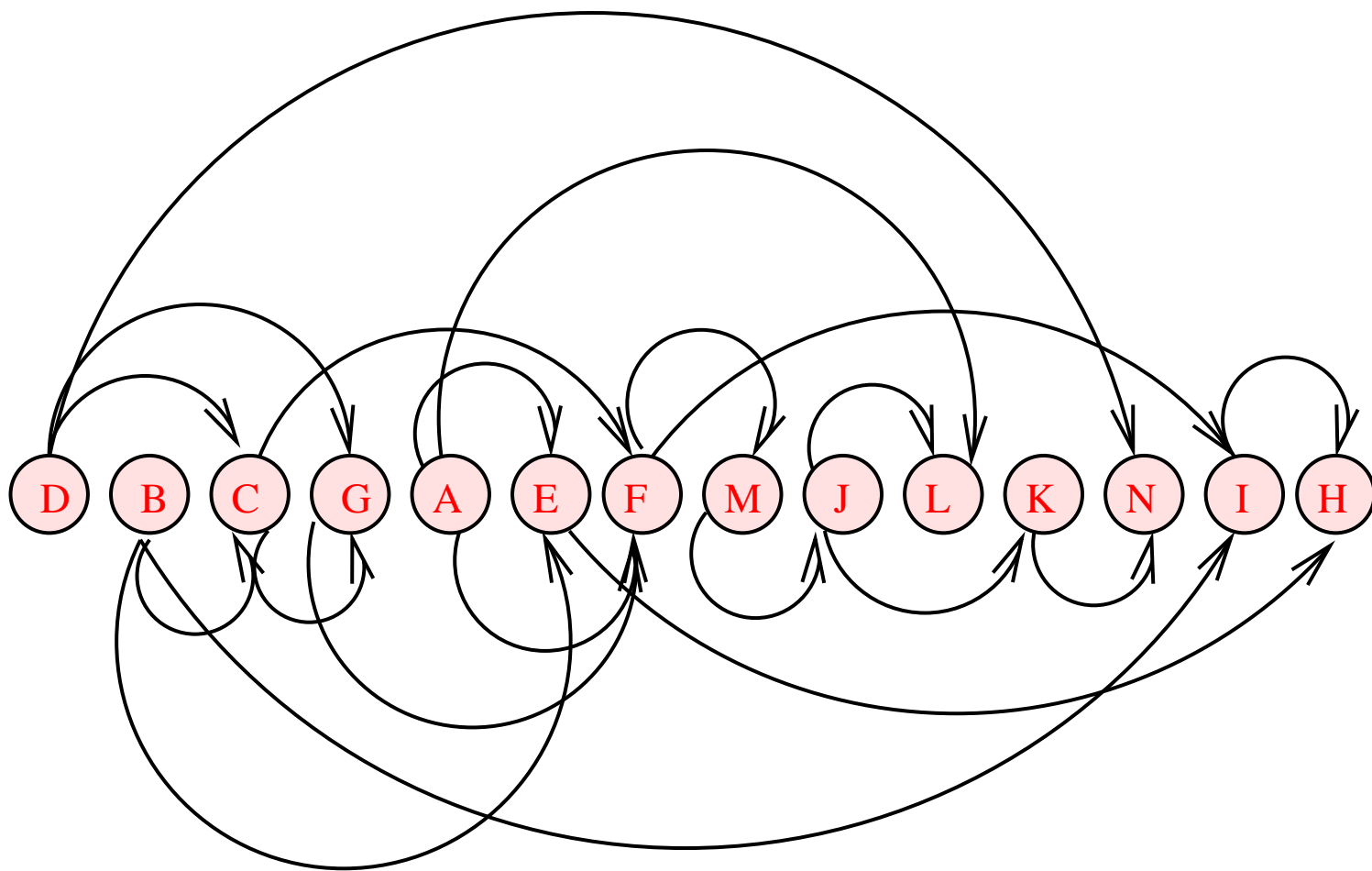


Kolejka: B, C, G, A, E, F, M, J, L, K, N, I, H



Kolejka: D, B, C, G, A, E, F, M, J, L, K, N, I, H





4. Spójność grafów

4.1 Składowe spójności

Definicja (Spójność grafu). Graf nieskierowany G jest spójny (słabo spójny) wtedy i tylko wtedy, gdy między każdą parą jego wierzchołków istnieje przynajmniej jedna ścieżka.

Definicja (Spójność grafu). Graf $G = (V, E)$ jest spójny wtedy i tylko wtedy, gdy dla każdego rozbicia V na niepuste podzbiory V_1 i V_2 istnieje krawędź z jednym końcem w V_1 a drugim w V_2 .

Definicja (Spójność wierzchołków, składowe spójności). Dany jest graf $G = (V, E)$. Dwa wierzchołki u i v ($u, v \in V$) nazywamy spójnymi jeżeli istnieje (u, v) -ścieżka w G .

Spójność wierzchołków jest relacją równoważności na zbiorze wierzchołków V . Zatem istnieje podział zbioru V na niepuste podzbiory (klasy równoważności) $V_1, V_2, \dots, V_\omega$ takie, że dwa wierzchołki są spójne wtedy i tylko wtedy, gdy należą do tego samego podzbioru V_i .

Podgrafy $G[V_1], G[V_2], \dots, G[V_\omega]$ nazywamy składowymi spójności grafu G , natomiast $\omega = \omega(G)$ oznacza liczbę składowych spójności grafu G .

- **DFS** i **BFS** systematycznie przechodzą wszystkie wierzchołki (i krawędzie) grafu, rozpoczynając od zadanego wierzchołka v
- mogą dojść tylko do wierzchołków znajdujących się w tej samej składowej co v
- można je znacznie uprościć, ponieważ podział krawędzi na podzbiory w tym zadaniu jest nieistotny

Algorithm 0.0.1: SKŁADOWADFS(G, v)

global $Składowa$ **comment:** Przed pierwszym wyw.: $Składowa \leftarrow \emptyset$

$Składowa \leftarrow Składowa \cup \{v\}$

for each $w \in \Gamma(v)$, $\Gamma(v) - \text{sasiedzi } v$

do if $w \notin Składowa$ **then** SKŁADOWADFS(G, w)

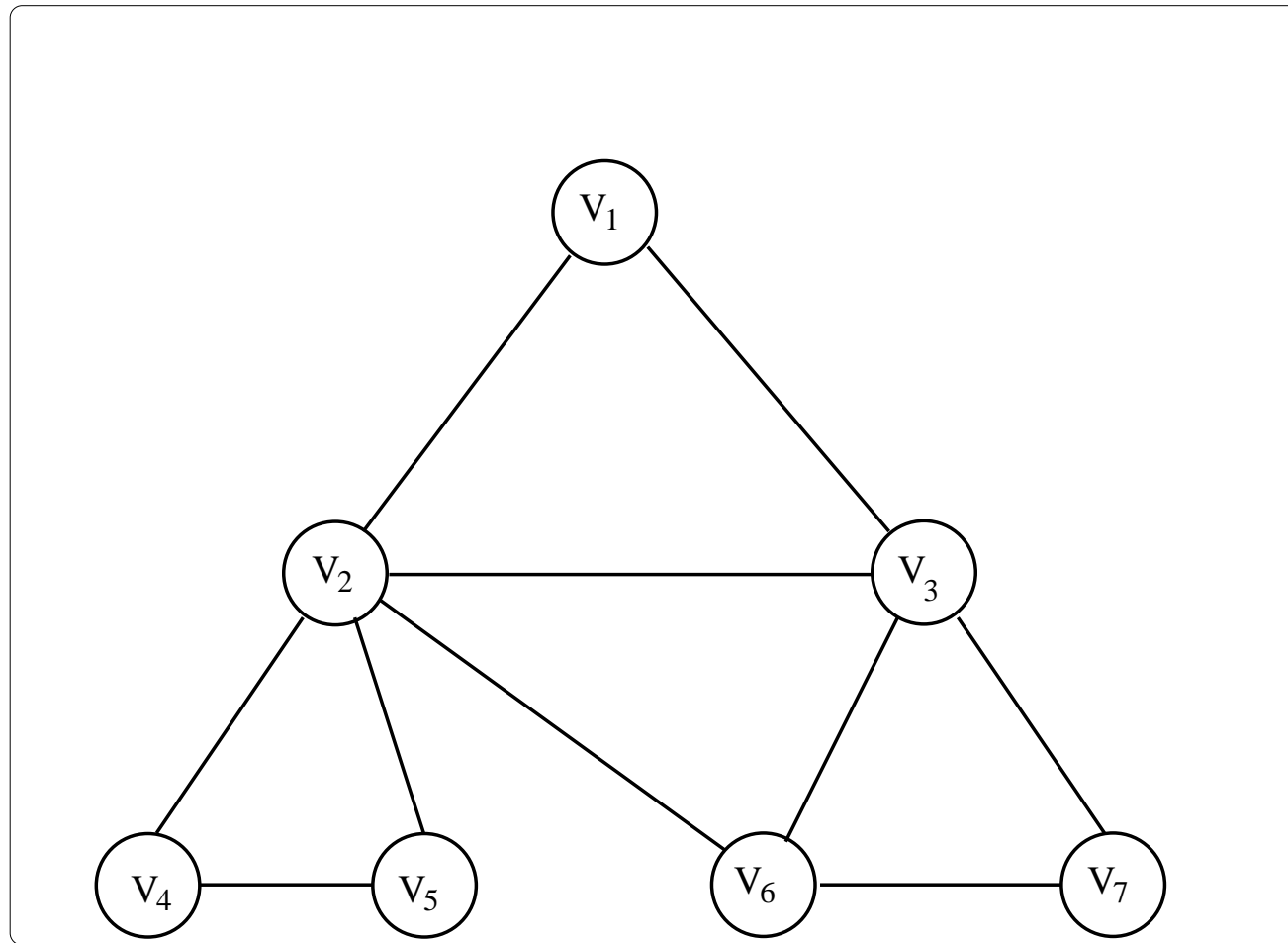
- w przypadku gdy interesuje nas jedynie **liczba składowych** grafu G to można użyć następującej procedury:

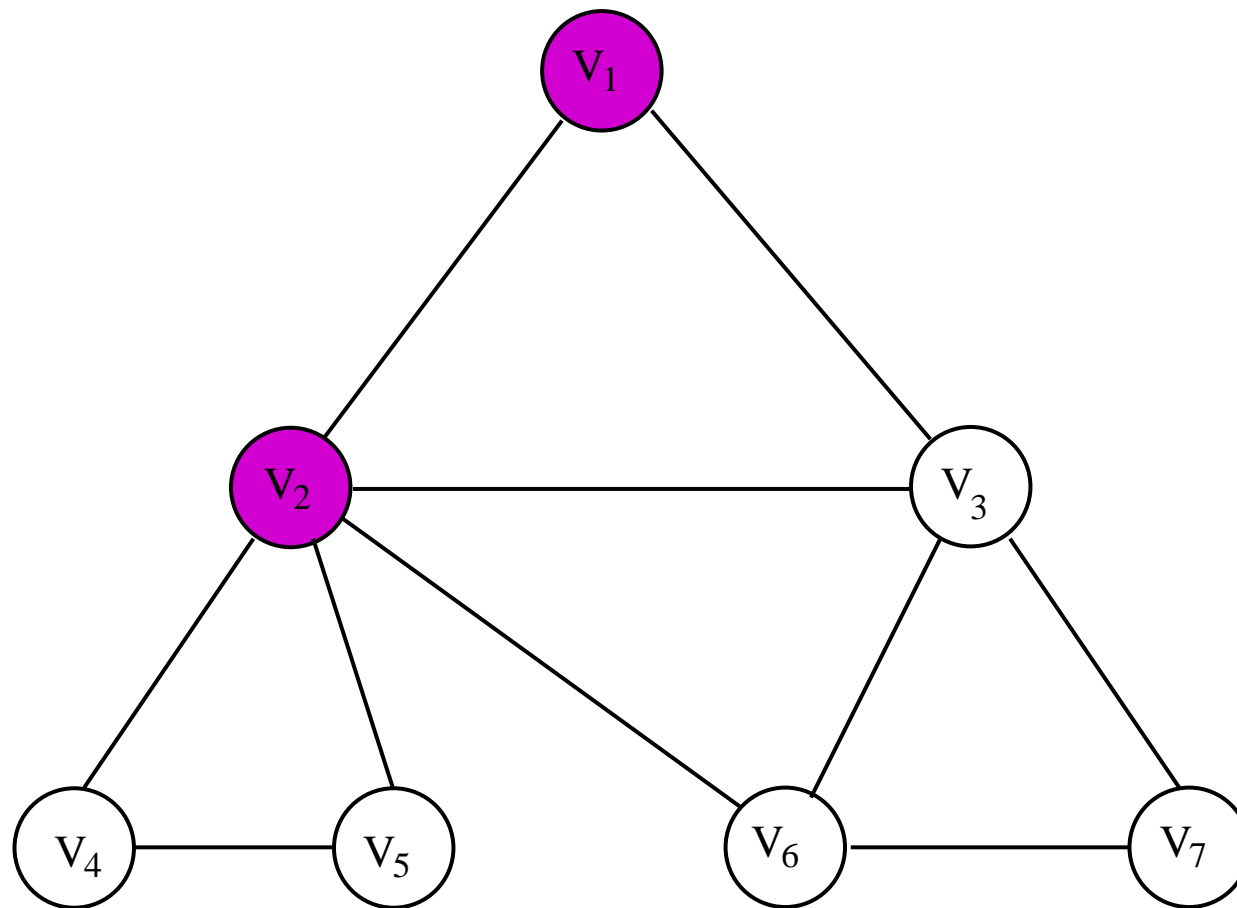
Algorithm 0.0.1: OMEGA(G)

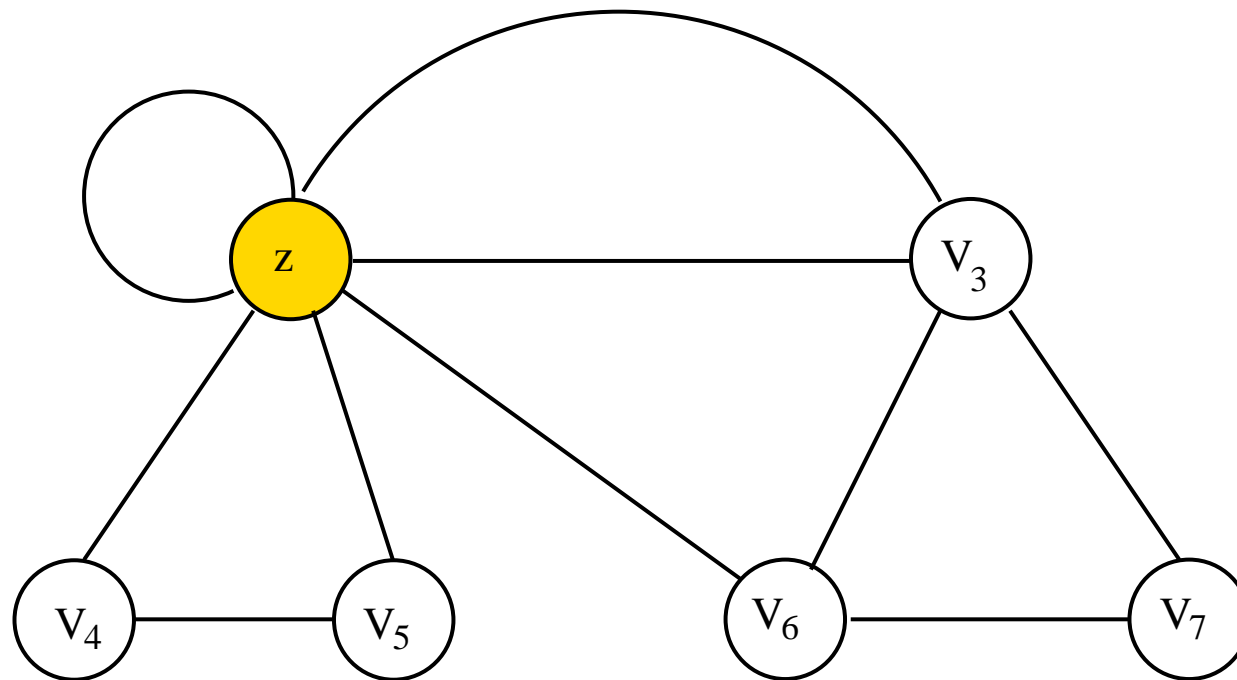
```
 $\omega \leftarrow 0$   
 $Składowa \leftarrow \emptyset$   
for each  $v \in V(G)$   
  do if  $v \notin Składowa$   
    then  $\begin{cases} SkładowaDFS(G, v) \\ \omega \leftarrow \omega + 1 \end{cases}$   
return  $(\omega)$ 
```

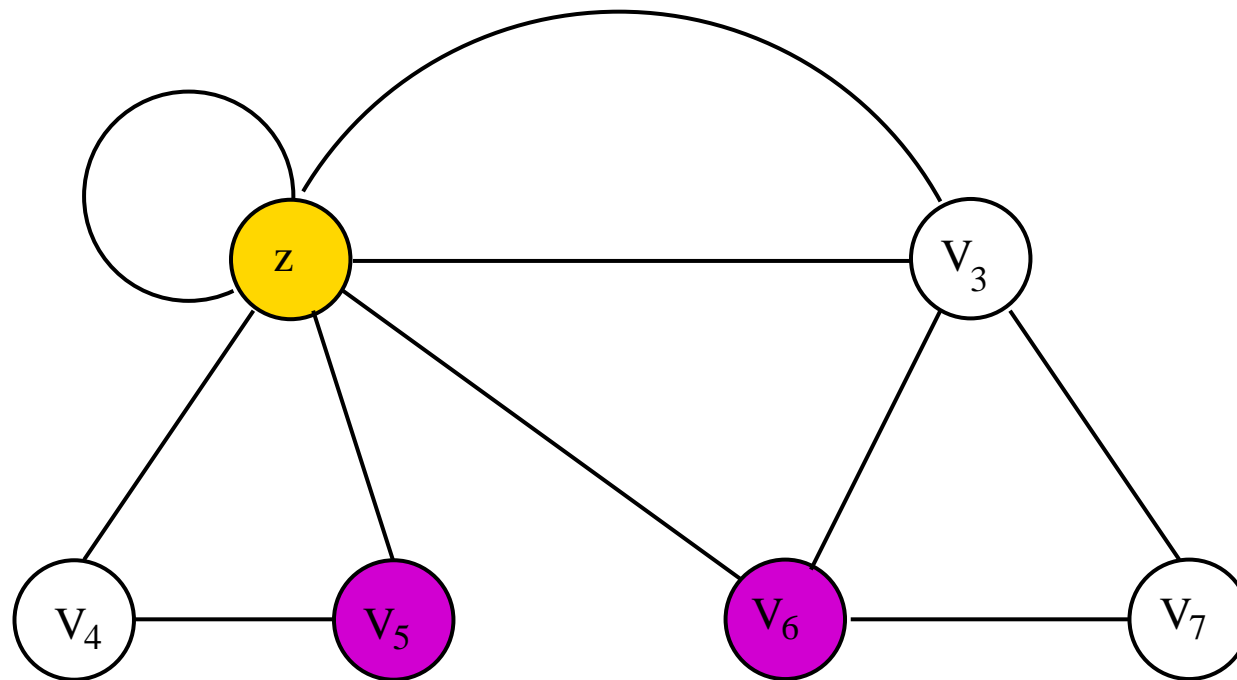
- złożoność obu powyższych algorytmów jest tego samego rzędu jak **DFS**, to znaczy $O(|V| + |E|)$

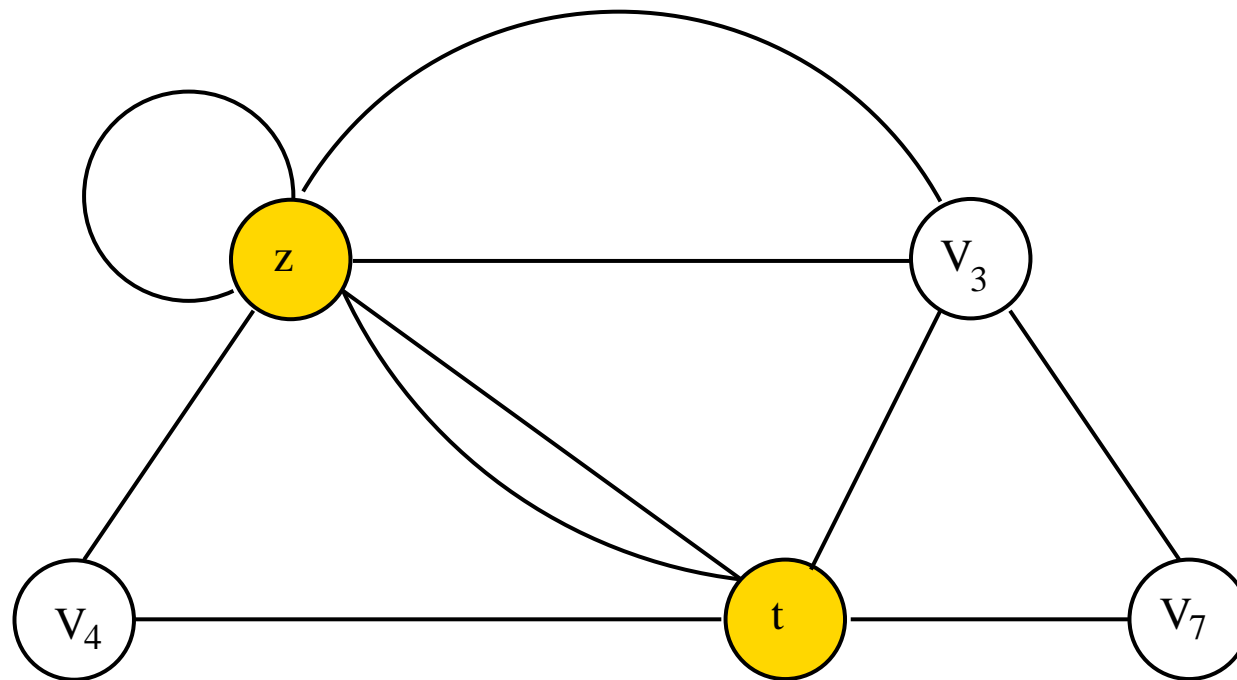
Algorytm scalania wierzchołków

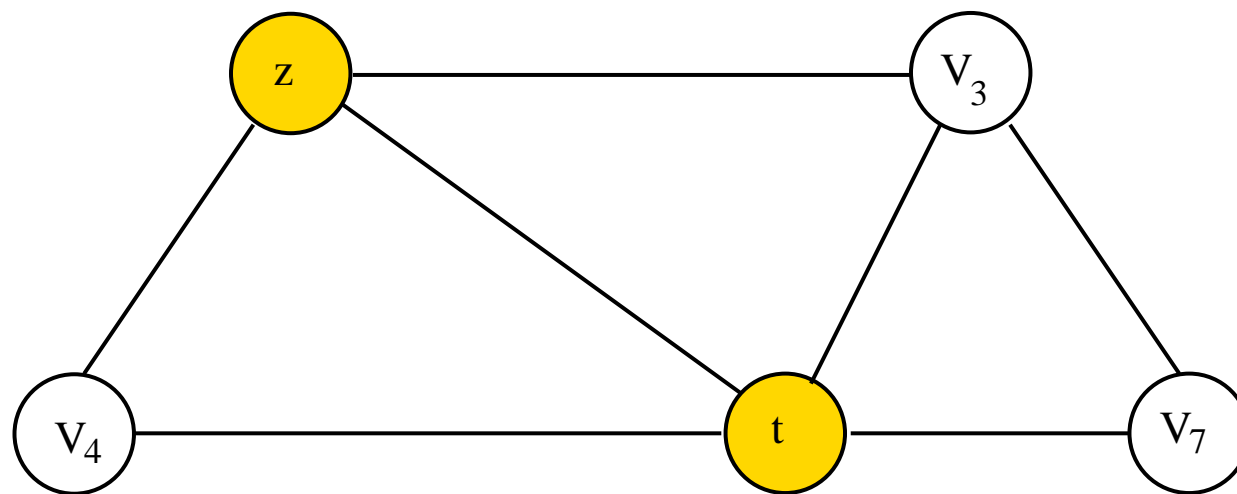












Niech v_i oraz v_j będą przyległymi wierzchołkami grafu $G = (V, E)$. Jeżeli graf $\hat{G} = (\hat{V}, \hat{E})$ jest otrzymany z grafu G przy pomocy operacji scalenia jego wierzchołków v_i oraz v_j , to spełnia on następujące warunki:

- $\hat{V} = V \setminus \{v_i, v_j\} \cup \{z\}$, gdzie $z \notin V$, tzn. zbiór wierzchołków \hat{V} „nowego” grafu otrzymujemy ze zbioru wierzchołków V wyjściowego grafu usuwając najpierw scalane wierzchołki v_i oraz v_j a następnie dodając do niego nowy wierzchołek z (powstały przez scalenie wierzchołków v_i, v_j).

- $(v_m, v_l) \in \hat{E}$, gdzie $(m, l \neq i, j)$ wtedy i tylko wtedy, gdy $(v_m, v_l) \in E$, tzn. wszystkie krawędzie grafu G , które nie są incydentne do żadnego z wierzchołków v_i ani v_j są również krawędziami „nowego” grafu \hat{G} .
- $(v_m, z) \in \hat{E}$, gdzie $(m \neq i, j)$ wtedy i tylko wtedy, gdy $(v_m, v_j) \in E$ lub $(v_m, v_i) \in E$; tzn. krawędź incydentna do wierzchołka z (powstałego przez scalenie wierzchołków v_i oraz v_j) w grafie \hat{G} odpowiada krawędzi incydentnej do wierzchołka v_i lub do v_j w grafie wyjściowym G (i na odwrót).

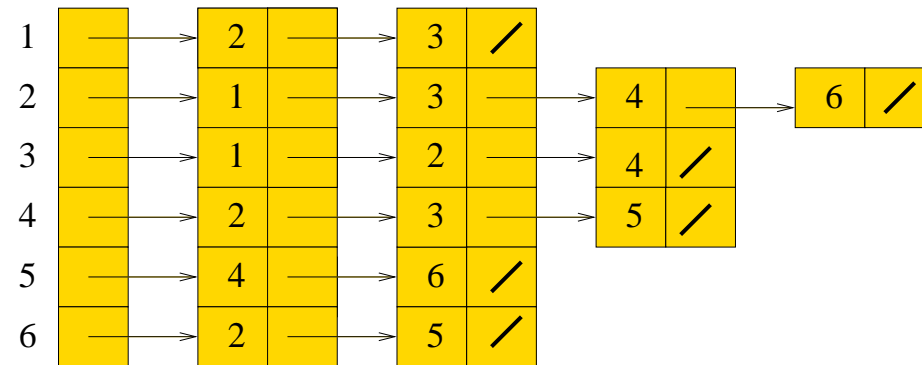
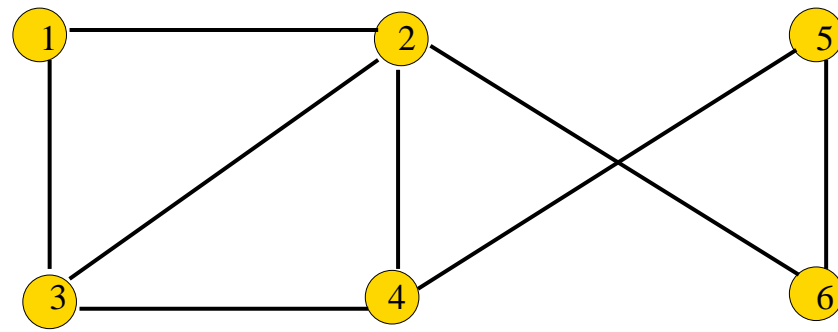
- $(z, z) \in \hat{E}$ wtedy i tylko wtedy, gdy $(v_i, v_i) \in E$ lub $(v_j, v_j) \in E$ lub $(v_i, v_j) \in E$; tzn., że pętla incydentna z nowo powstałym wierzchołkiem z w grafie \hat{G} powstaje wtedy i tylko wtedy, gdy w grafie G istniała pętla incydentna z wierzchołkiem v_j lub z wierzchołkiem v_i , lub istniała krawędź (v_i, v_j) .
- w operacji scalania można zastąpić wielokrotne krawędzie pojedynczą i usunąć pętle
- scalenie dwóch przyległych wierzchołków nie zmienia liczby składowych spójności

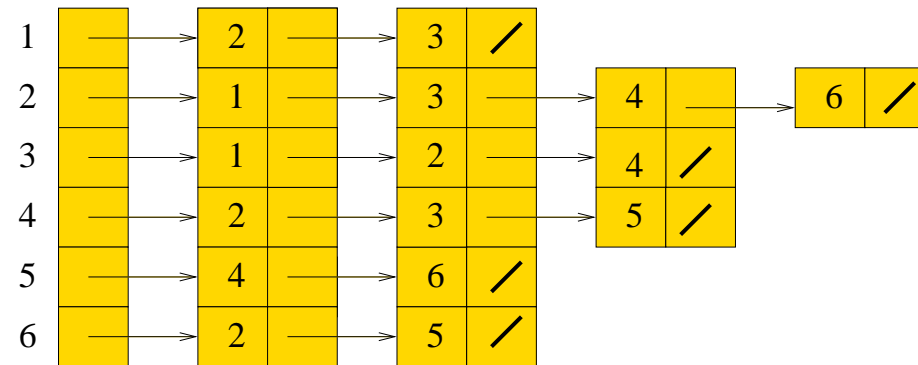
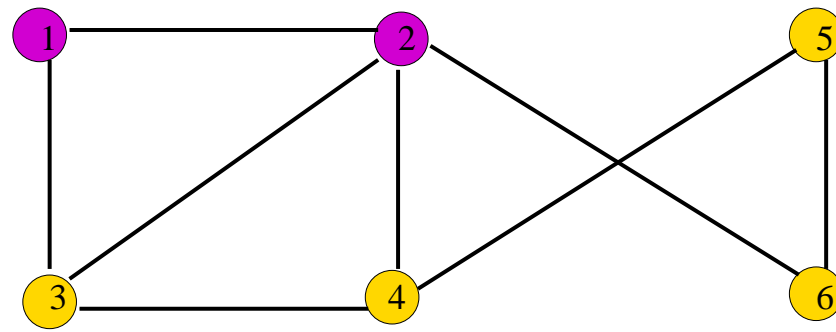
Opis algorytmu scalania

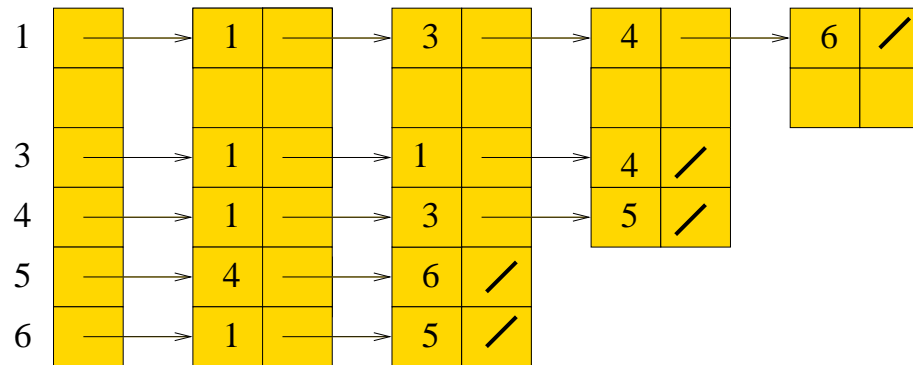
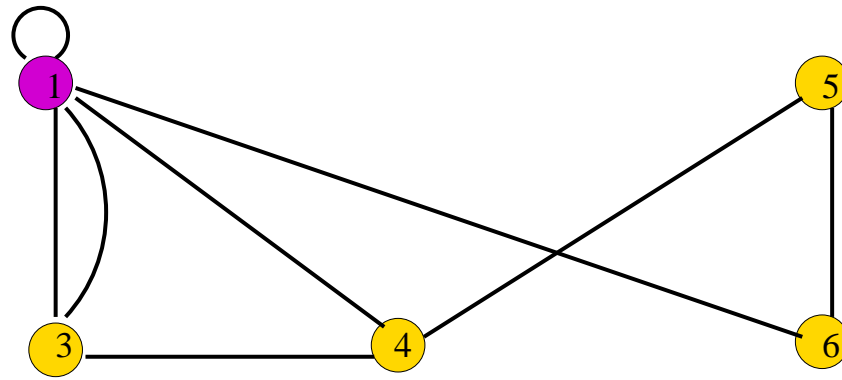
- dany jest graf G : rozpoczynamy od pewnego wierzchołka w grafie i scalamy kolejno wszystkie wierzchołki przyległe do niego
- następnie bierzemy scalony wierzchołek i znów kolejno scalamy z nim wszystkie te wierzchołki, które są teraz do niego przyległe
- proces scalania powtarzamy do momentu, gdy scalony wierzchołek jest izolowany. To wskazuje, że pewna spójna składowa została „scalona” do pojedynczego wierzchołka

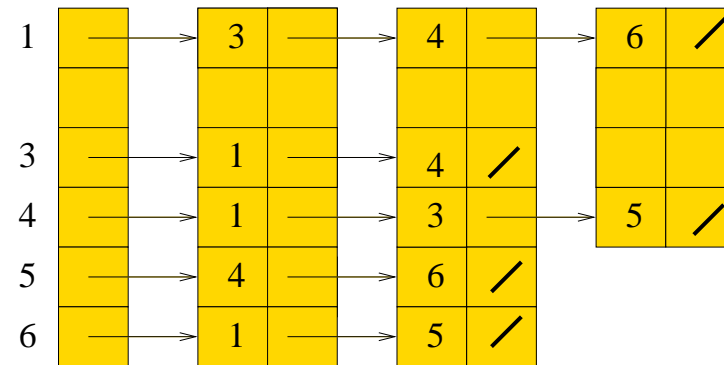
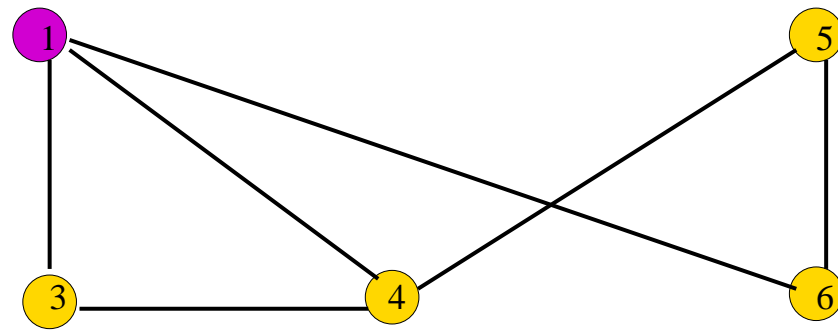
- jeżeli poza „scalonym” wierzchołkiem nie ma już innych wierzchołków w grafie, to graf jest spójny
- w przeciwnym przypadku, możemy analogicznie wyznaczyć pozostałe wierzchołki: usuwamy wierzchołek otrzymany w wyniku scalania (zapisujemy zbiór wierzchołków, z których on powstał jako kolejną składową spójności) i rozpoczynamy naszą procedurę scalania od dowolnego wierzchołka w otrzymanym grafie

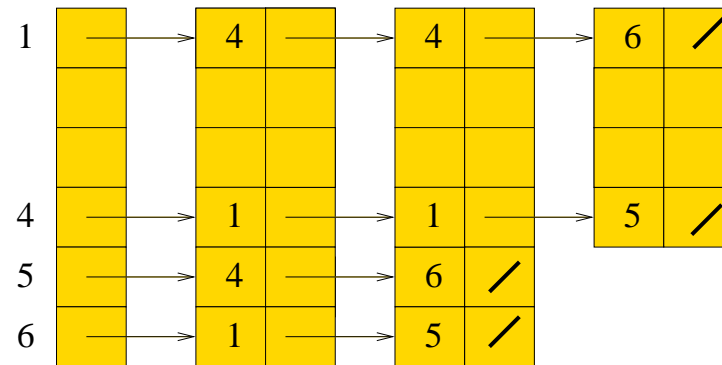
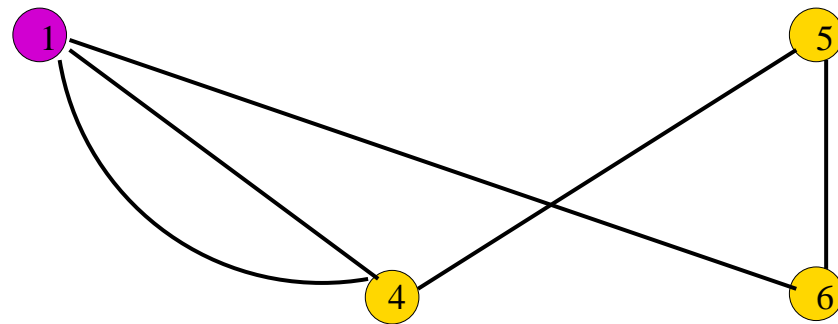
- operację scalania wierzchołków łatwo jest wykonać dla typowych reprezentacji komputerowych grafów
- w przypadku listy krawędzi usuwamy listy odpowiadające wierzchołkom v_i i v_j , a następnie tworzymy nową listę dla wierzchołka z będącą sumą usuniętych list (dla wierzchołków v_i i v_j). Należy pamiętać o zamianie na wszystkich listach elementów v_i oraz v_j na z .

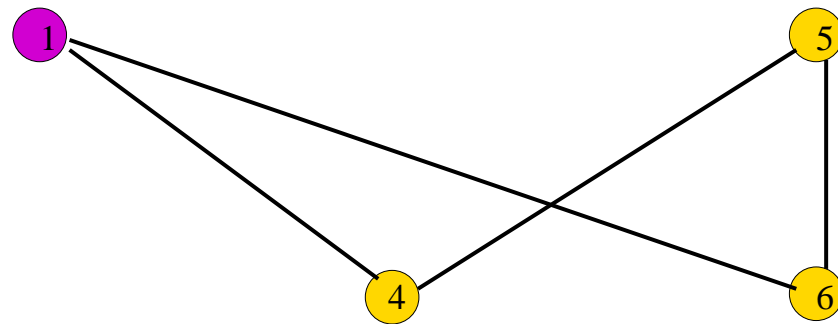




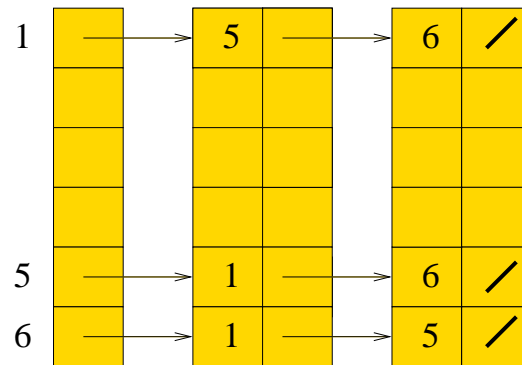
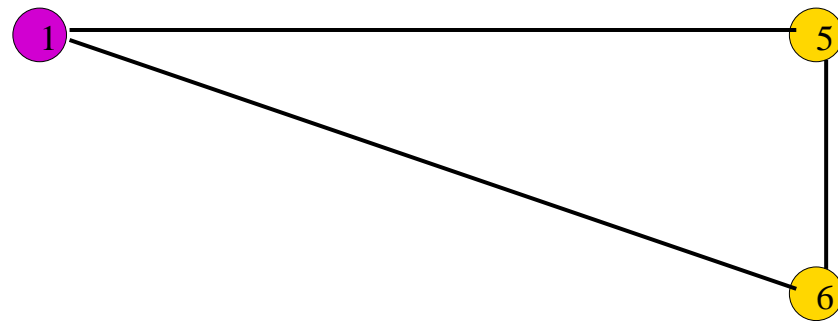


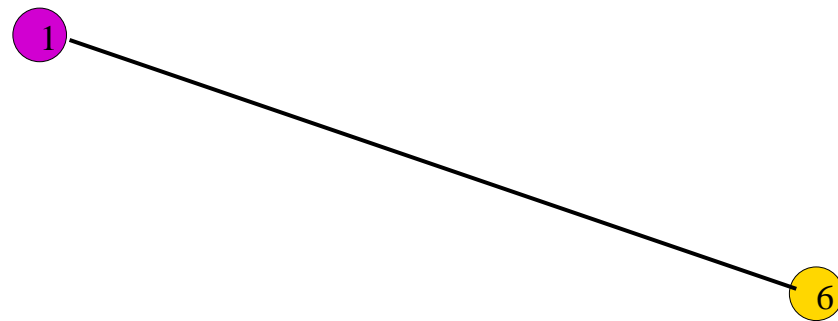






1	→	4	→	6	/
4	→	1	→	5	/
5	→	4	→	6	/
6	→	1	→	5	/





1		→	6	/
6		→	1	/

1

1



- w macierzy przyległości A scalanie wierzchołka j -tego z i -tym dokonuje się za pomocą operacji \vee (dodawania logicznego) j -tego wiersza do i -tego wiersza oraz j -tej kolumny do i -tej kolumny. Następnie j -ty wiersz i j -ta kolumna są usuwane z macierzy
- w macierzy przyległości zamiana i -tej i j -tej kolumny z równoczesną zamianą i -tego i j -tego wiersza odpowiada zmianie numeracji wierzchołków, więc możemy założyć, że zawsze będziemy scalać wierzchołek pierwszy z i -tym i wynikowy scalony wierzchołek będzie pierwszym wierzchołkiem

Algorytm SCALWIERZCHOŁKI scala wierzchołek pierwszy z i -tym. Parametr A oznacza macierz przyległości, a n liczbę wierzchołków. Aby usunąć wierzchołek v_i algorytm wpisuje na jego miejsce ostatni wierzchołek (czas $O(n)$ zamiast $O(n^2)$ w przypadku przesuwania elementów tablicy).

Zwróćmy uwagę, że algorytm nie sprawdza, czy scalane wierzchołki są przyległe.

Algorithm 0.0.1: SCALWIERZCHOŁKI(A, n, i)

```
for  $k \leftarrow 1$  to  $n$ 
  do  $\begin{cases} A[1, k] \leftarrow A[1, k] \vee A[i, k] \\ A[k, 1] \leftarrow A[k, 1] \vee A[k, i] \\ A[1, k] \leftarrow A[n, k] \\ A[k, 1] \leftarrow A[k, n] \end{cases}$ 
 $n \leftarrow n - 1$ 
return ( $A, n$ )
```

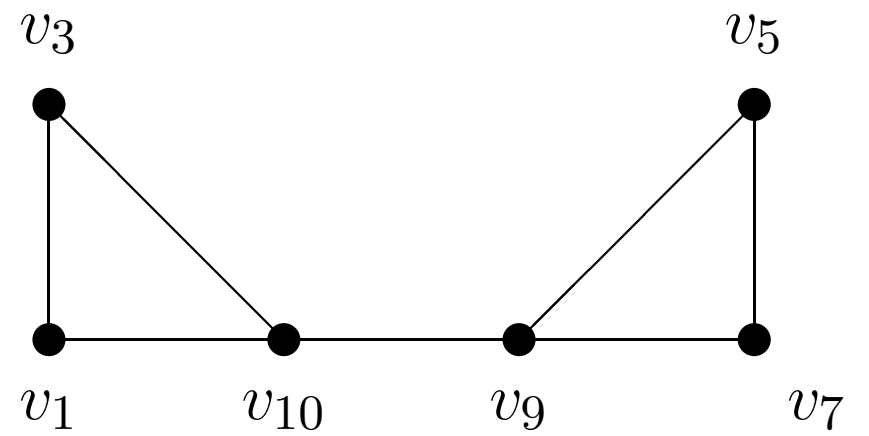
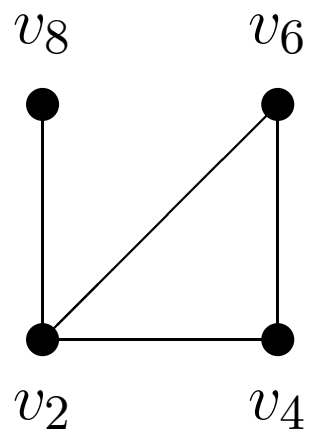
- do wyznaczenia składowej spójności zawierającej wierzchołek v możemy wykorzystać algorytm scalania wierzchołków SCALWIERZCHOŁKI, jak to ma miejsce w przedstawionym poniżej algorytmie SCALSKAŁDOWA
- zauważmy, że algorytm SCALSKAŁDOWA zamienia najpierw wierzchołek v z pierwszym, a następnie tak długo scala wierzchołki aż wierzchołek 1 stanie się izolowany
- ponieważ algorytm zmienia numerację wierzchołków, więc tablica $perm$ służy do odtworzenia numeracji pierwotnej, dokładniej $perm[i]$ jest numerem wierzchołka odpowiadającego i -tej kolumnie macierzy A .

Algorithm 0.0.1: SCALSKŁADOWA(A, n, v)

```
global  $perm$ 
for  $k \leftarrow 1$  to  $n$ 
  do  $\begin{cases} A[1, k] \leftrightarrow A[v, k] \\ A[k, 1] \leftrightarrow A[k, v] \end{cases}$ 
 $perm[v] \leftarrow perm[1]$ 
 $z \leftarrow \{v\}$ 
repeat
   $i \leftarrow 2$ 
  while  $(i < n) \wedge (A[1, i] = 0)$ 
    do  $i \leftarrow i + 1$ 
  if  $A[1, i] = 0$ 
    then return  $(A, n, z)$ 
  else  $\begin{cases} z \leftarrow z \cup \{perm[i]\} \\ \text{SCALWIERZCHOŁKI}(A, n, i) \\ perm[i] \leftarrow perm[n + 1] \end{cases}$ 
until false
```


Przykład . Znaleźć składową słabej spójności zawierającą wierzchołek v_8 grafu danego macierzą przyległości:

$$A(G) = \begin{matrix} & \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \end{matrix} & \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{matrix} .$$

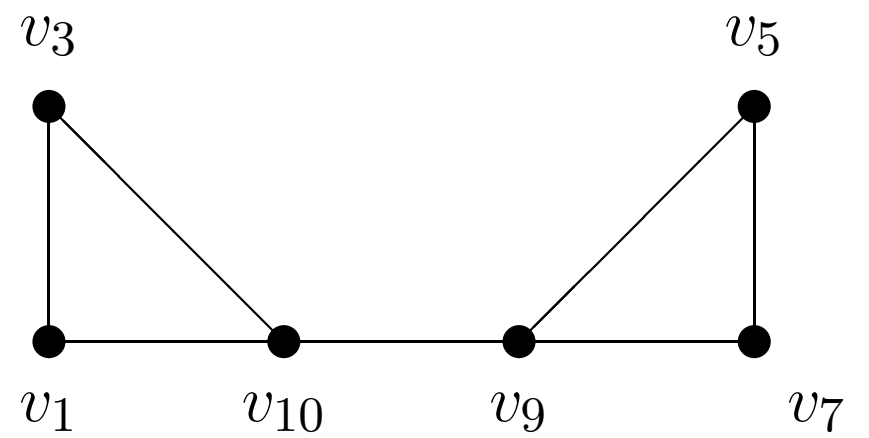
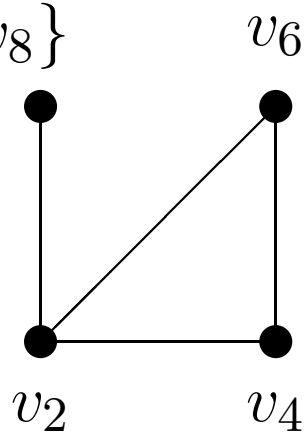


W pierwszym kroku zamieniamy miejscami wierzchołki v_8 i v_1 oraz inicjujemy zmienne:

$$A(G) = \begin{array}{c} perm \\ z \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_1 \\ v_9 \\ v_{10} \end{array} \begin{pmatrix} z & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_1 & v_9 & v_{10} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix},$$

$$z = \{v_8\}.$$

$$z = \{v_8\}$$

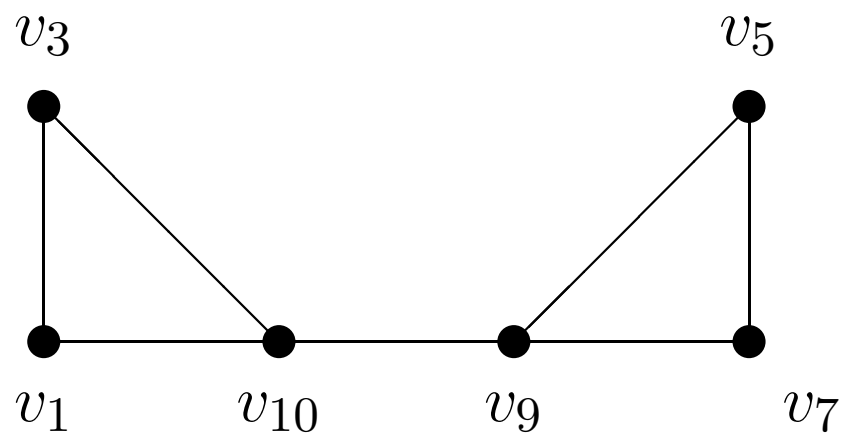
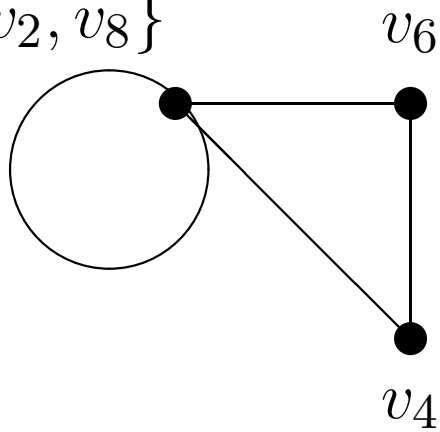


Ponieważ wierzchołek z jest incydentny z v_2 więc dokonujemy ich scalenia poprzez $\text{SCALWIERZCHOŁKI}(A, 10, 2)$. W wyniku otrzymujemy:

$$A(G) = \begin{matrix} perm & z & v_{10} & v_3 & v_4 & v_5 & v_6 & v_7 & v_1 & v_9 \\ \begin{matrix} z \\ v_{10} \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_1 \\ v_9 \end{matrix} & \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{matrix},$$

$$z = \{v_2, v_8\}.$$

$$z = \{v_2, v_8\}$$

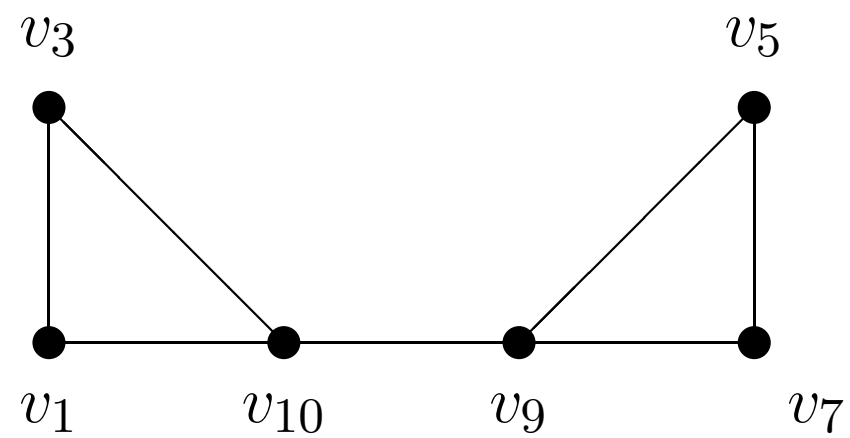
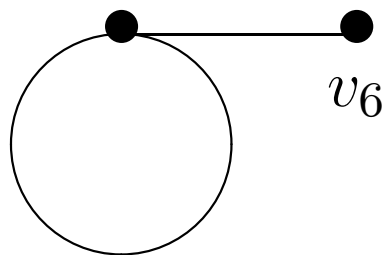


Tym razem pierwszym incydentnym do z wierzchołkiem jest v_4 , więc skalamy je poprzez $\text{SCALWIERZCHOŁKI}(A, 9, 4)$ i otrzymujemy:

$$A(G) = \begin{matrix} perm & z & v_{10} & v_3 & v_9 & v_5 & v_6 & v_7 & v_1 \\ \begin{matrix} z \\ v_{10} \\ v_3 \\ v_9 \\ v_5 \\ v_6 \\ v_7 \\ v_1 \end{matrix} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix},$$

$$z = \{v_2, v_4, v_8\}.$$

$$z = \{v_2, v_4, v_8\}$$

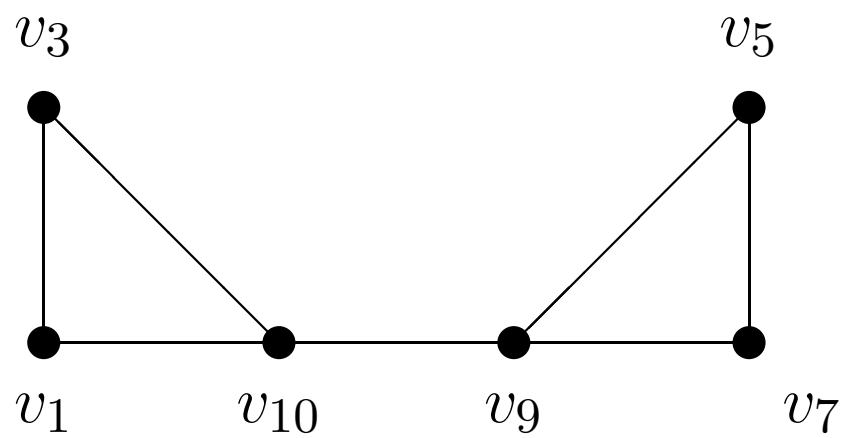
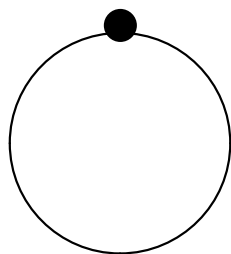


Następnie scalamy wierzchołki z i v_6 :

$$A(G) = \begin{array}{c} perm \\ z \\ v_{10} \\ v_3 \\ v_9 \\ v_5 \\ v_1 \\ v_7 \end{array} \begin{pmatrix} z & v_{10} & v_3 & v_9 & v_5 & v_1 & v_7 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix},$$

$$z = \{v_2, v_4, v_6, v_8\}.$$

$$z = \{v_2, v_4, v_6, v_8\}$$



4.1 Składowe silnej spójności digrafów

Definicja (Osiągalność wierzchołków). Dany jest graf skierowany (digraf) $G = (V, E)$. Mówimy że wierzchołek $w \in V$ jest osiągalny z wierzchołka $v \in V$ jeżeli istnieje skierowana (v, w) -ścieżka w grafie G .

Definicja (Macierz osiągalności). Macierzą osiągalności grafu skierowanego $G = (V, E)$ nazywamy macierz zerojedynkową $D(G) = (d_{ij})$ ($1 \leq i, j \leq \nu(G)$) której element d_{ij} jest równy jedynce, gdy wierzchołek v_j jest osiągalny z wierzchołka v_i , a zeru w przypadku przeciwnym.

Definicja (Wzajemna osiągalność wierzchołków). Mówimy, że wierzchołki v i w digrafu $G = (V, E)$ są wzajemnie osiągalne jeżeli w grafie G wierzchołek v jest osiągalny z w a w jest osiągalny z v .

Definicja (Silna składowa spójności). Podgraf indukowany $G[\hat{V}]$ grafu skierowanego $G = (V, E)$ nazywa się silną składową spójności (fragmentem grafu) jeżeli każda para wierzchołków z \hat{V} jest wzajemnie osiągalna w $G[\hat{V}]$. W szczególności jeżeli każda para wierzchołków z V jest wzajemnie osiągalna to graf jest silnie spójny.

Twierdzenie . *Każdy maksymalny podzbiór wierzchołków, którym odpowiadają identyczne wiersze macierzy osiągalności, tworzy składową silnej spójności.*

Dowód. Elementy głównej przekątnej macierzy osiągalności $D(G)$ są z definicji jedynkami. Jeżeli zatem weźmiemy dowolne dwa identyczne wiersze macierzy $D(G)$, na przykład wiersze i -ty i j -ty, to

$$d_{ij} = d_{ii} = 1 = d_{jj} = d_{ji} .$$

Oznacza to, że wierzchołki v_i oraz v_j są wzajemnie osiągalne i co za tym idzie należą do tej samej składowej silnej spójności. Biorąc zatem wszystkie wierzchołki, którym odpowiadają identyczne wiersze, otrzymujemy odpowiednią składową silnej spójności. ■

- czy powyższe twierdzenie rozwiązuje problem badania silnej spójności grafu?
- niestety wyznaczanie macierzy osiągalności jest pracochłonne
- przypomnijmy, że w macierzy $P^k(G)$ będącej k -tą potęgą binarnej **macierzy przejść** $P(G)$ digrafu G (odpowiednika macierzy przyległości $A(G)$ dla grafu G) element o współrzędnych (i, j) jest **liczbą** (v_i, v_j) -ścieżek długości k w grafie G
- jeżeli jednak operacje wykonamy nad pierścieniem **algebry Boole'a** to element ten jest równy 1 wtedy i tylko wtedy, gdy istnieje (v_i, v_j) -ścieżek długości k .

Macierz osiągalności można wyznaczyć za pomocą następującego algorytmu:

1. Utwórz macierz binarną $B = P(G) + I$.
2. Dla $k = 1, 2, \dots$, gdzie $k \leq n - 1$ obliczaj B^k (nad algebrą Boole'a) tak długo, aż $B^k = B^{k-1}$.

Można łatwo wykazać, że $B^k = D(G)$. Niestety, może się zdażyć, że będziemy musieli wyznaczyć wszystkie potęgi, aż do $k = n - 1$, co dla dużych grafów może być bardzo pracochłonne (mimo, że złożoność tego algorytmu jest wielomianowa względem n).

Inny prosty algorytm wyznaczania $D(G)$:

1. Wierzchołek v_i otrzymuje cechę $c = i$.
2. Wybieramy dowolny ocechowany wierzchołek v_k i rozpatrujemy odpowiadający mu k -ty wiersz w macierzy $P(G)$. Wszystkim nieocechowanym wierzchołkom v_j , dla których $p_{k,j} = 1$ nadajemy cechę $c = i$. Procedurę punktu 2 kontynuujemy dotąd, aż nie będzie można ocechować żadnych nowych wierzchołków.
3. W wierszu i -tym macierzy $D(G)$ wstawiamy jedynki na pozycjach odpowiadających wierzchołkom ocechowanym cechą $c = i$.

BFS !! BFS !! BFS !!

Algorytm Leifmana (1966)

Oznaczmy półstopnie wierzchołka v skierowanego grafu $G = (V, E)$:

$$d_+(v) = |\{(v_1, v_2) \in E : v_1 = v\}|,$$

$$d_-(v) = |\{(v_1, v_2) \in E : v_2 = v\}|.$$

W opisie algorytmu stosujemy następujące oznaczenia:

$l(v)$ — lewa cecha wierzchołka v ,

$p(v)$ — prawa cecha wierzchołka v ,

\mathcal{S} — zbiór podzbiorów wierzchołków tworzących składowe silnej spójności (**fragmenty**) digrafu.

1. W grafie G poszukujemy wierzchołków v , dla których $d_+(v) = 0$ lub $d_-(v) = 0$. Jeżeli są takie wierzchołki, to zaliczamy je do \mathcal{S} , jako jednowierzchołkowe składowe silnej spójności. Usuwamy je z grafu G , tworząc odpowiedni podgraf $G^o = (V^o, E^o)$ i dla tego podgrafu kontynuujemy krok 1 tak długo, aż nie będzie już takich wierzchołków.
2. W podgrafie $G^o = (V^o, E^o)$ (uzyskanym w wyniku czynności kroku 1 lub 4) wybieramy dowolny wierzchołek v_o i rozpoczynamy proces cechowania wierzchołków podwójnymi cechami $(l(v), p(v))$, który realizujemy w następujący sposób:

- (a) Na początku wszystkim wierzchołkom $v \in V^o$ nadajemy wartości cech $(0, 0)$. Wszystkim następnikom v wierzchołka v_o nadajemy wartość cechy $l(v) = 1$. Nieocechowanym następnikom tych ocechowanych wierzchołków też nadajemy wartość cechy $l = 1$ itd., aż dojdziemy do sytuacji, w której nie będzie można powiększyć zbioru wierzchołków z cechą $l = 1$. W wyniku tego postępowania, cechę $l = 1$ mają wszystkie wierzchołki osiągalne z wierzchołka v_o .
- (b) Podobnie, startując znów z wierzchołka v_o , lecz poruszając się przeciwnie do zwrotu łuków, nadajemy prawe cechy $p(v) = 1$ wierzchołkom v , z których jest osiągalny wierzchołek v_o za pomocą ścieżek (skierowanych) o niezerowej długości.

Komentarz: Po zakończeniu tego etapu każdy wierzchołek $v \in V^o$ ma nadane wartości (zero lub jeden) obu cech $l(x)$ i $p(x)$. W ten sposób zbiór V^o został podzielony na cztery rozłączne podzbiory

$$V^o = V_{00} \cup V_{01} \cup V_{10} \cup V_{11} ,$$

gdzie indeksy oznaczają odpowiednio wartości cech (lewej i prawej) wierzchołków należących do tych podzbiorów.

Algorytm Leifmana c.d.

3. Określamy nowe składowe silnej spójności. Możliwe są przy tym następujące sytuacje:

- (a) $V_{11} = \emptyset$. Wtedy zbiór V_{00} zawiera wierzchołek v_o , tworzący jednowierzchołkową składową silnej spójności $\{v_o\}$. Zapamiętujemy zbiór $V_{00} \setminus \{v_o\}$, jeżeli jest on niepusty, wraz z pozostałymi niepustymi zbiorami spośród V_{01}, V_{10} . Przechodzimy do kroku 4.
- (b) $V_{11} \neq \emptyset$. Wtedy V_{11} stanowi kolejną, składową silnej spójności, którą zaliczamy do zbioru \mathcal{S} . Jeżeli pozostałe zbiory V_{00}, V_{01}, V_{10} są puste, to przechodzimy do kroku 4. W przeciwnym przypadku zapamiętujemy niepuste zbiory i będziemy je oddzielnie rozpatrywali w następnych etapach procedury, ponieważ każda nie wyznaczona do tej pory składowa silnej spójności zawiera się całkowicie w jednym z tych zbiorów. Przechodzimy do kroku 4.

4. Pytamy: czy są zapamiętane nierozpatrzone zbiory powstałe w wyniku realizacji kroku 3? Jeżeli są, to wybieramy dowolny z nich i traktując go jako nowy zbiór V^o , tworzymy podgraf $G^o = \langle V^o, E^o \rangle$, z którym przechodzimy do kroku 2. Jeżeli nie ma już zapamiętanych i nierozpatrzonych zbiorów, to zbiór \mathcal{S} zawiera wszystkie podzbiory tworzące składowe silnej spójności. Oznacza to koniec algorytmu Leifmana.

Pseudokod tego algorytmu przedstawiono jako Algorytm LEIFMAN(G). Dane to graf G a wynik to rodzina zbiorów \mathcal{S} . W algorytmie tym, dla uproszczenia zapisu, zasymulowano stos tablicą. W praktycznych zastosowaniach należy jednak użyć odpowiedniej struktury dynamicznej.

Algorithm 0.0.1: LEIFMAN(G)

```

 $\mathcal{S} \leftarrow \emptyset$ ;  $Stos[1] \leftarrow V(G)$ ;  $NStos \leftarrow 1$ 
repeat
   $V_o \leftarrow Stos[NStos]$ ;  $NStos \leftarrow NStos - 1$ 
  repeat
     $znaleziono \leftarrow 0$ 
    for each  $v \in V_o$ 
      do if  $d_+(v) = 0 \vee d_-(v) = 0$ 
        then  $\begin{cases} znaleziono \leftarrow v; \mathcal{S} \leftarrow \mathcal{S} \cup \{\{v\}\}; V_o \leftarrow V_o \setminus \{v\}; \\ G \leftarrow G - v \\ \text{break} \end{cases}$ 
    until  $znaleziono = 0$ 
    if  $V_o \neq \emptyset$ 
      then  $\begin{cases} \text{for each } v \in V_o \\ \text{do } l(v) \leftarrow p(v) \leftarrow 0 \\ \text{wybierz dowolne } v_o \in V_o \\ S[1] \leftarrow v_o; NS \leftarrow 1 \\ \text{repeat} \\ v \leftarrow s[NS]; NS \leftarrow NS - 1 \\ \text{for each } w \in \Gamma_+(v) \\ \text{do if } l(w) = 0 \\ \text{then } l(w) \leftarrow 1; NS \leftarrow NS + 1; S[NS] \leftarrow w \\ \text{until } NS = 0 \\ S[1] \leftarrow v_o; NS \leftarrow 1 \\ \text{repeat} \\ v \leftarrow s[NS]; NS \leftarrow NS - 1 \\ \text{for each } w \in \Gamma_-(v) \\ \text{do if } p(w) = 0 \\ \text{then } p(w) \leftarrow 1; NS \leftarrow NS + 1; S[NS] \leftarrow w \\ \text{until } NS = 0 \end{cases}$ 
     $V_{00} \leftarrow V_{01} \leftarrow V_{10} \leftarrow V_{11} \leftarrow \emptyset$ 
    for each  $v \in V_o$ 
      do  $V_{l(v),p(v)} \leftarrow V_{l(v),p(v)} \cup \{v\}$ 
    if  $V_{11} \neq \emptyset$  then  $\mathcal{S} \leftarrow \mathcal{S} \cup \{V_{11}\}$ ;  $G \leftarrow G - V_{11}$ 
    for each  $V_{ij}$ 
      do if  $(i, j) \neq (0, 0) \wedge V_{ij} \neq \emptyset$ 
        then  $NStos \leftarrow NStos + 1$ ;  $STos[NStos] \leftarrow V_{ij}$ 
    until  $NStos = 1$ 
return ( $\mathcal{S}$ )

```

Dla $l, p = 1, 2, 3, 4$ oznaczmy przez $G_{lp} = G[V_{lp}]$ podgraf generowany przez podzbiór wierzchołków V_{lp} , będący jednym z czterech podzbiorów uzyskanych w wyniku procedury cechowania wierzchołków podgrafu G^o , w punkcie 2 algorytmu Leifmana.

Twierdzenie 1. *Jeżeli V_{11} jest zbiorem pustym, to wierzchołek początkowy v_o stanowi jednowierzchołkową składową silnej spójności i należy do zbioru V_{00} .*

Twierdzenie 2. *Jeżeli $V_{11} \neq \emptyset$, to podgraf G_{11} jest składową silnej spójności, a wierzchołek v_o , od którego zaczyna się cechowanie wierzchołków, należy do V_{11} .*

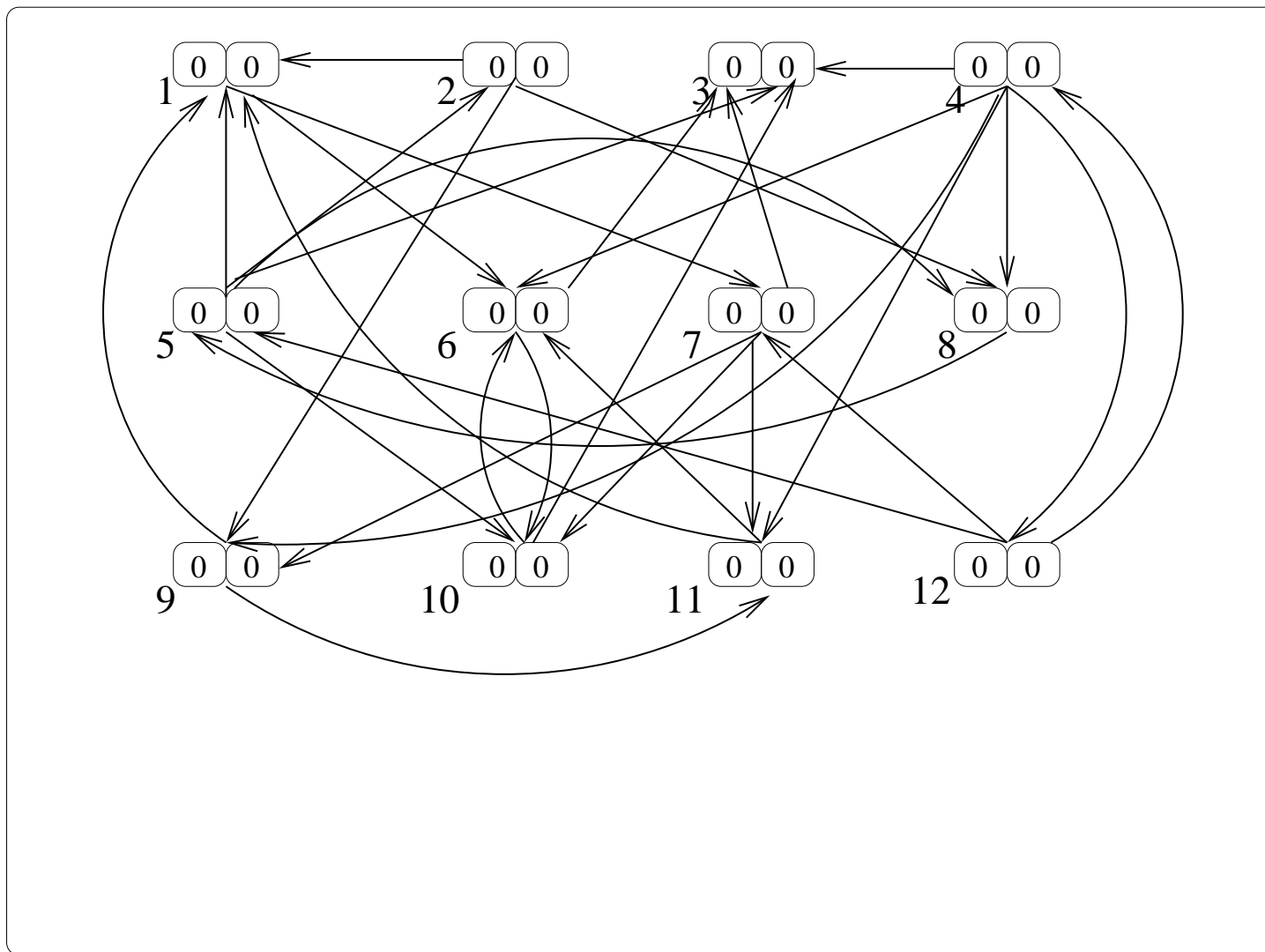
Twierdzenie 3. *Każda składowa silnej spójności grafu G^o zawiera się całkowicie w jednym z czterech podgrafów $G_{lp} = G[V_{lp}]$.*

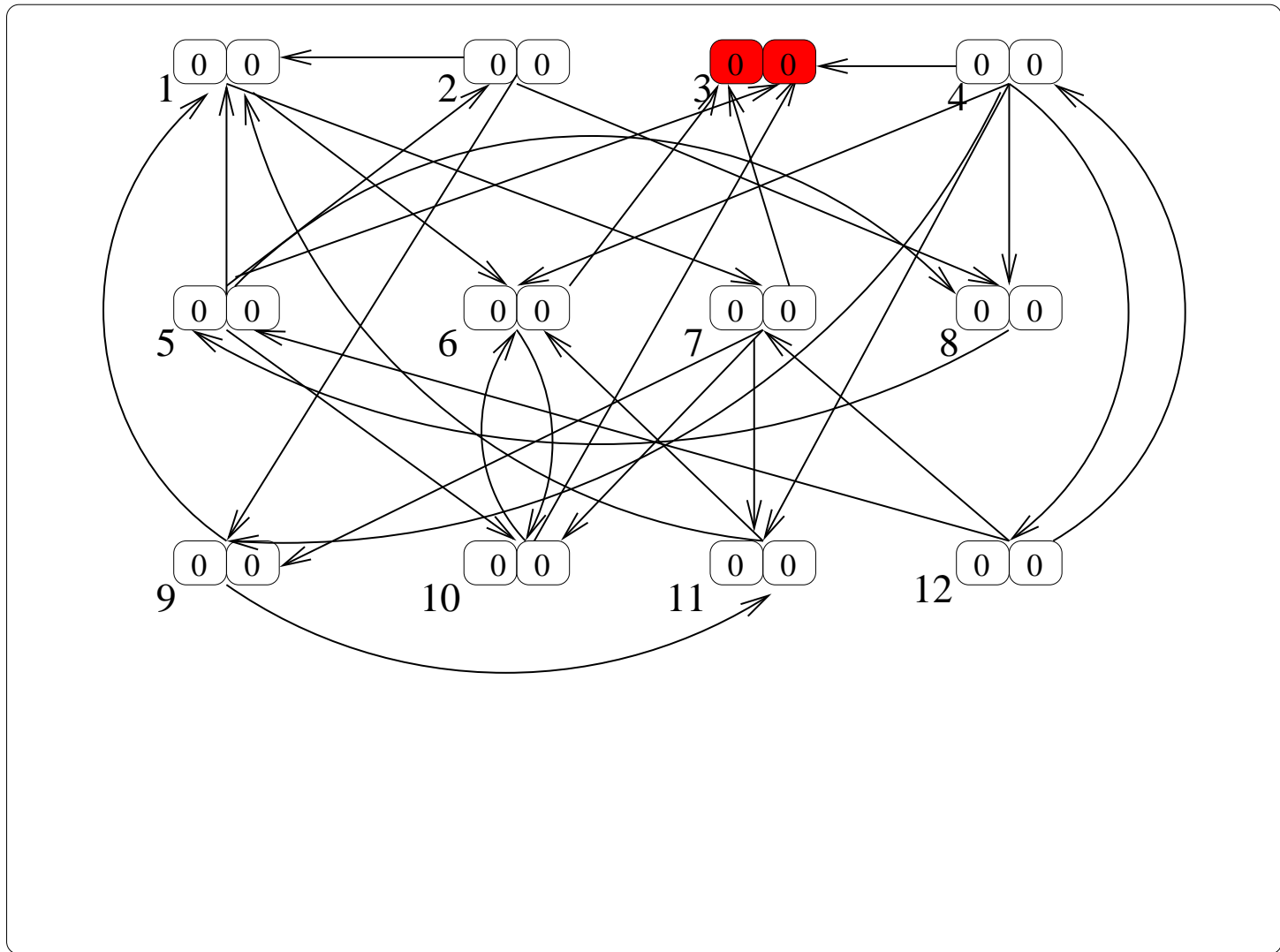
Dowód. Niech zbiór $U \subset V^o$ stanowi zbiór wierzchołków składowej silnej spójności grafu G^o . Weźmy dowolny wierzchołek $u \in U$. Wierzchołek v_o jest początkowym wierzchołkiem procedury cechowania. Jeżeli w grafie G^o istnieją (v_o, u) - i (u, v_o) -ścieżki, to na mocy Twierdzenia 2 zbiór $U = V_{11}$. Pozostaje przypadek, gdy nie istnieją jednocześnie obie (v_o, u) - i (u, v_o) -ścieżki. Można wtedy wyróżnić trzy sytuacje:

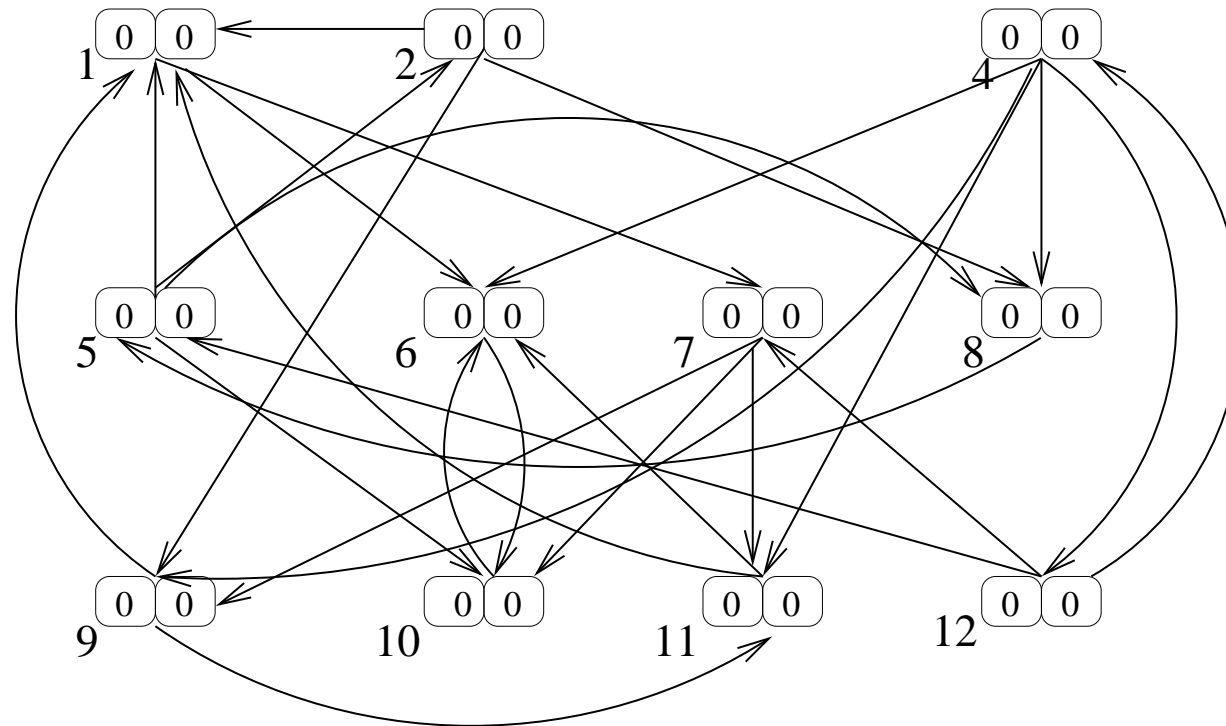
1. Istnieje (v_o, u) -ścieżka i nie istnieje (u, v_o) -ścieżka.
Wtedy wierzchołek u otrzymuje wartość cechy $l = 1$, $p = 0$. Tym samym wszystkie wierzchołki należące do U na mocy definicji silnej spójności i z przechodniości relacji osiągalności muszą mieć wartości cechy $l = 1$ i $p = 0$, a zatem $U \subset V_{10}$.
2. Nie istnieje (v_o, u) -ścieżka, a istnieje (u, v_o) -ścieżka.
Wtedy wierzchołek u otrzymuje cechę $l = 0$, $p = 1$.
Zatem wszystkie wierzchołki należące do U otrzymują te same wartości cech i cały zbiór U należy do V_{01} .
3. Nie istnieje ani (v_o, u) - ani (u, v_o) -ścieżka. Wtedy wszystkie wierzchołki zbioru U otrzymują wartości cech $l = 0$, $p = 0$ i tym samym cały zbiór U należy do V_{00} .

Przykład . Wyznaczyć za pomocą algorytmu Leifmana wszystkie składowe silnej spójności grafu skierowanego o macierzy przejść:

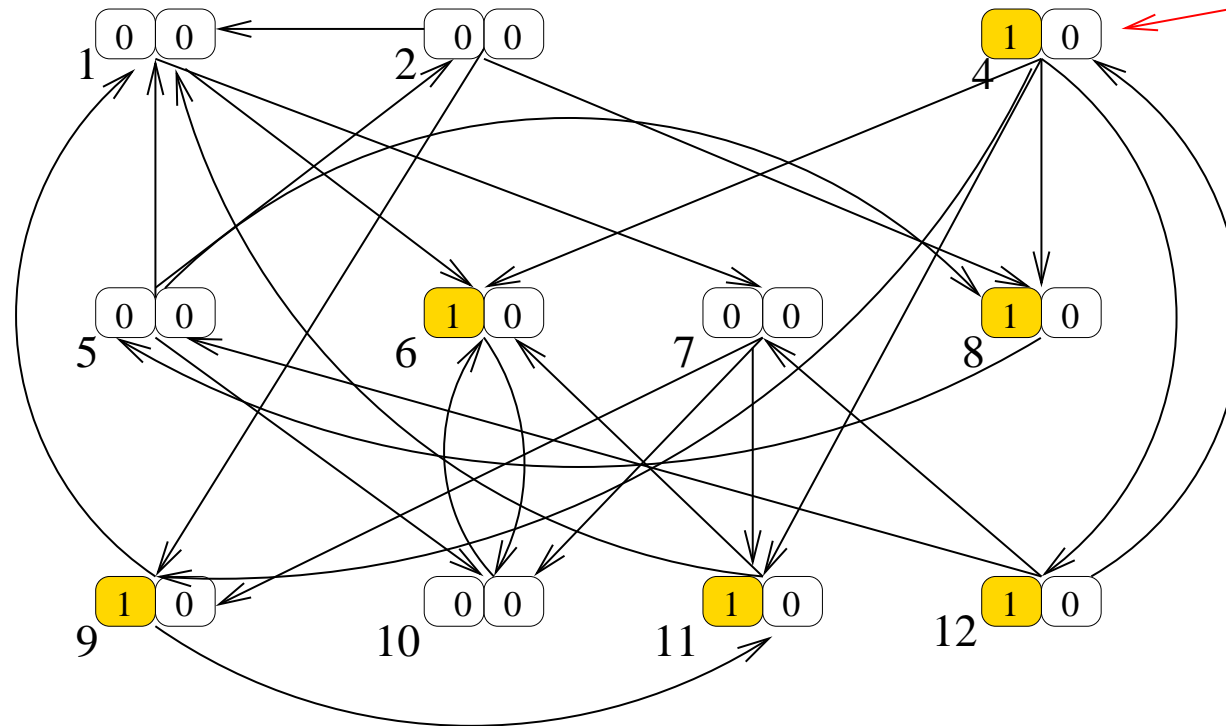
$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



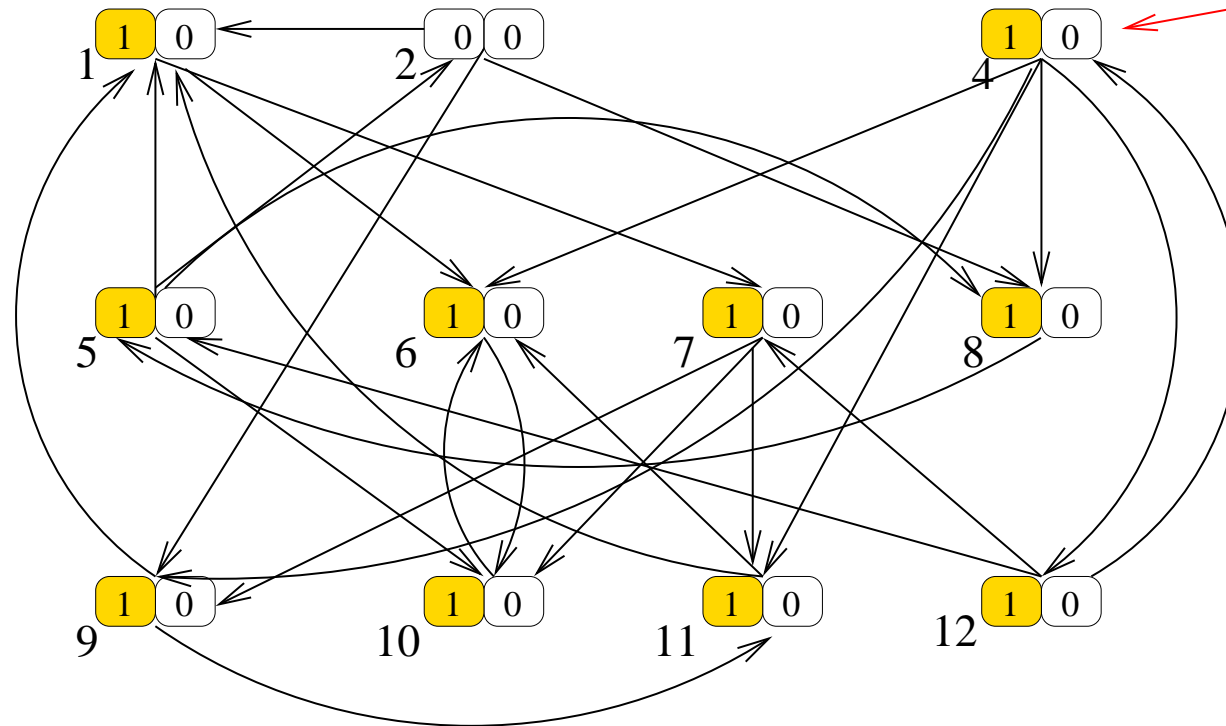




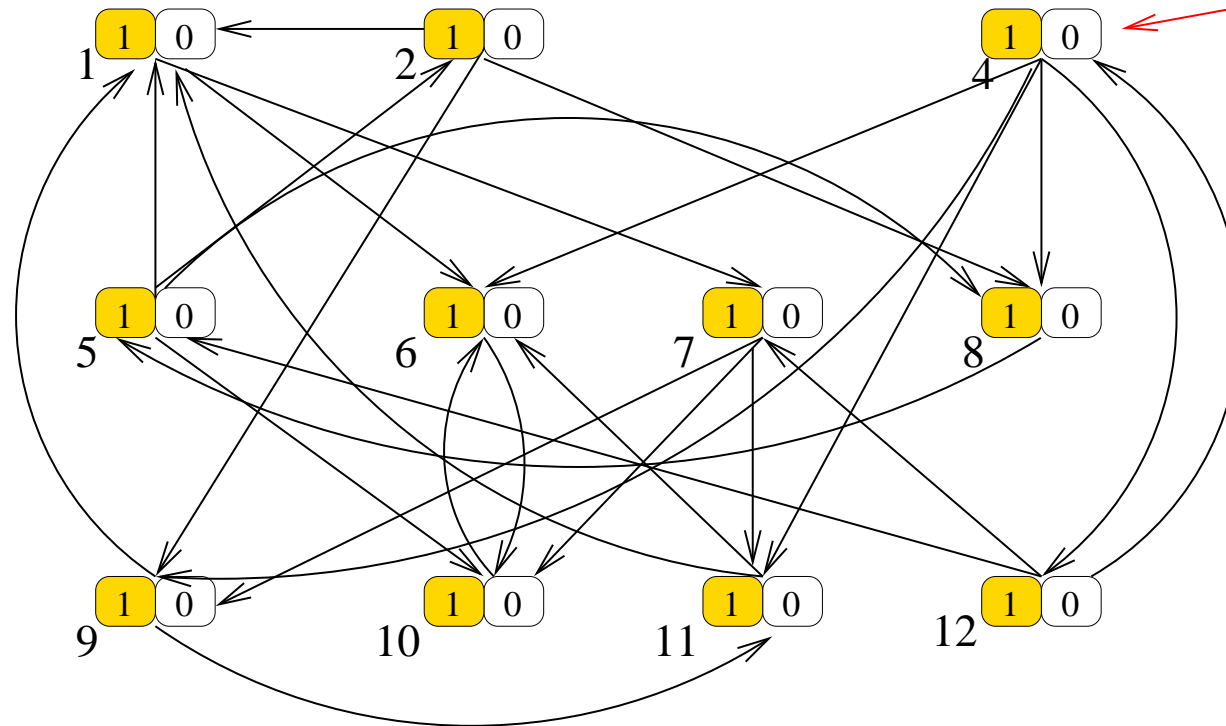
$$S_1 = \{3\}$$



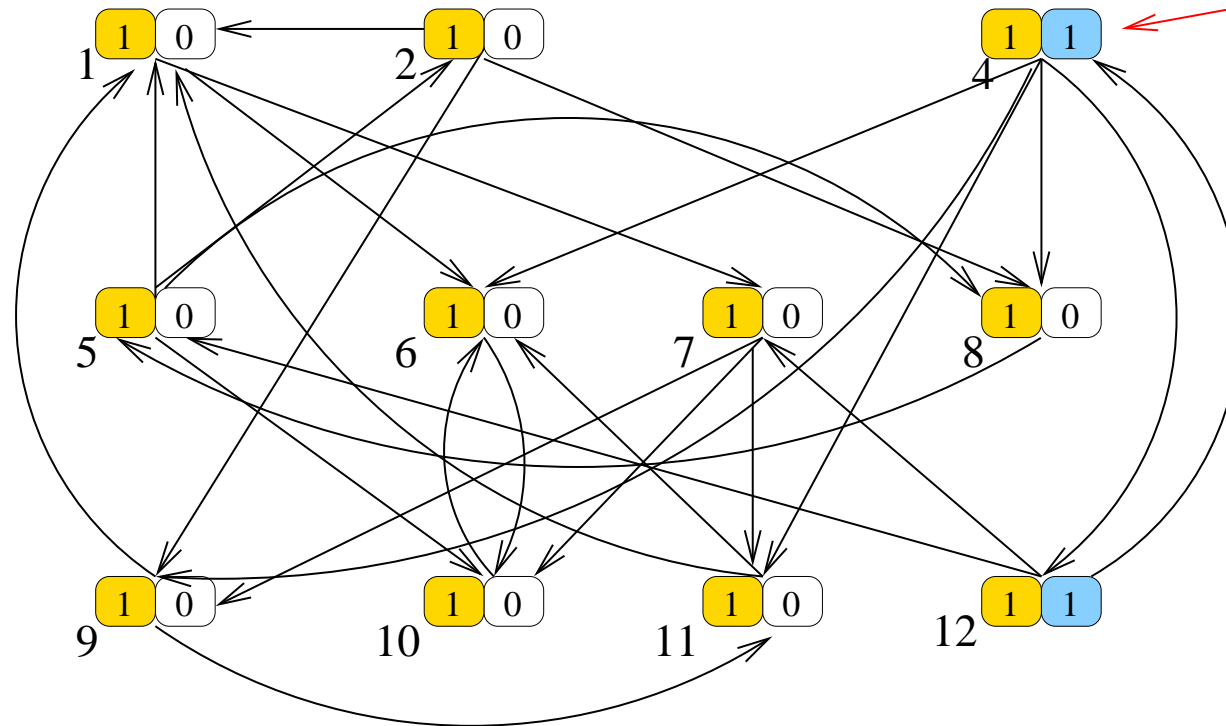
$S_1 = \{3\}$



$$S_1 = \{3\}$$

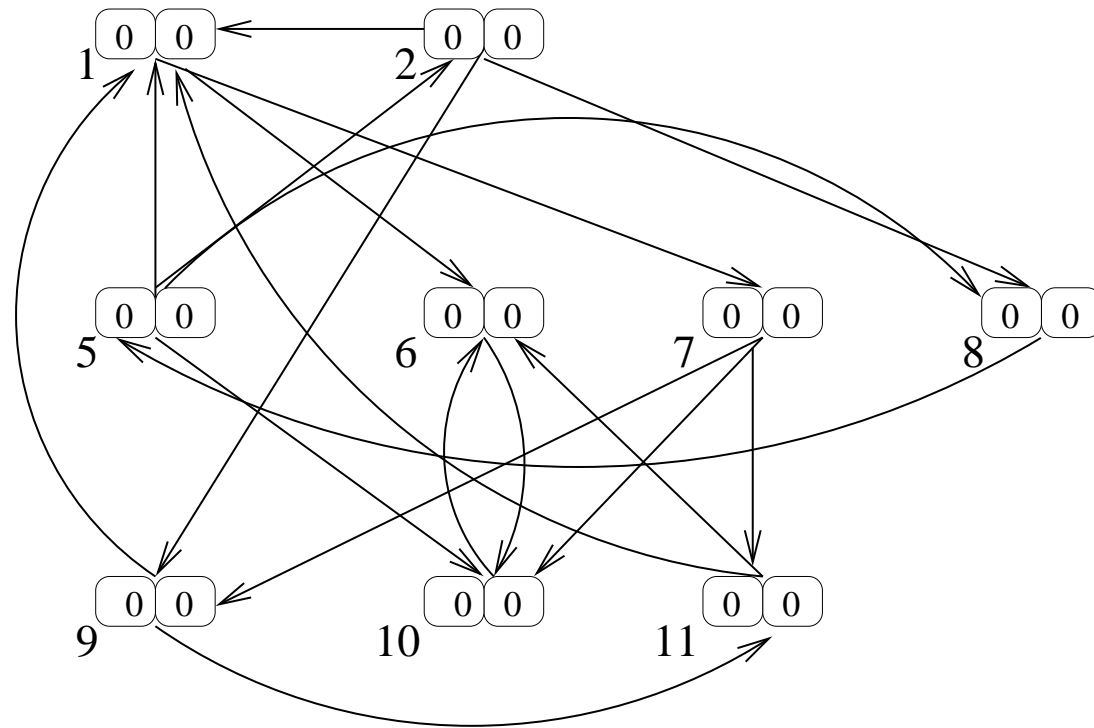


$$S_1 = \{3\}$$

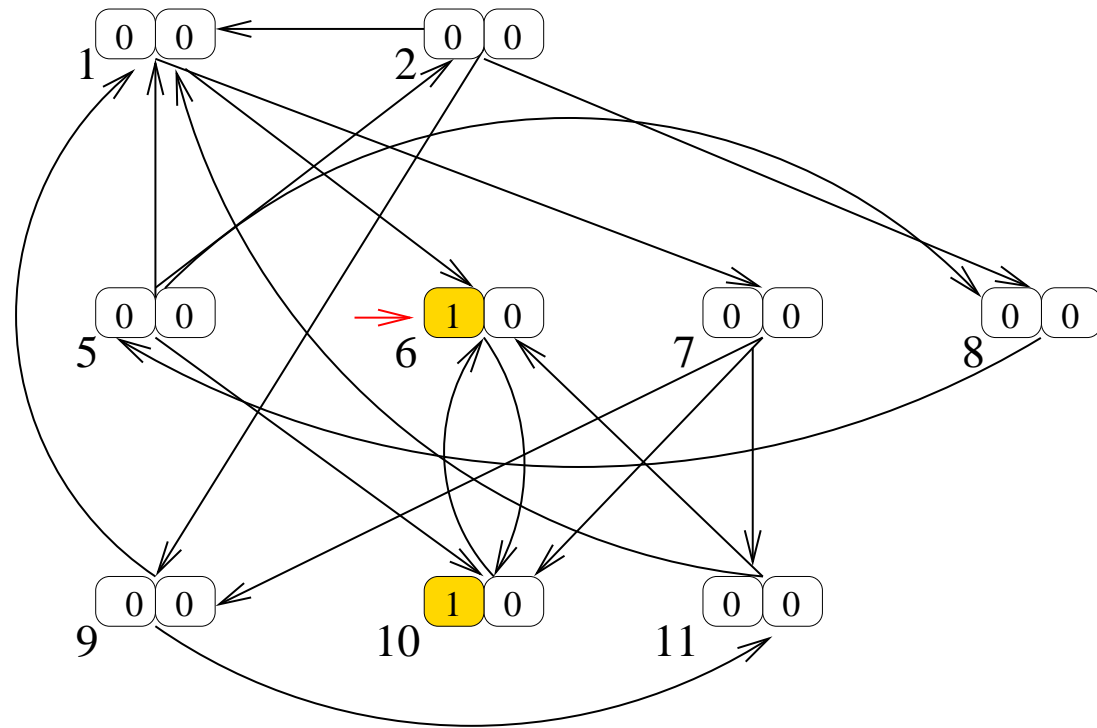


$$S_1 = \{3\} \quad S_2 = \{4, 12\}$$

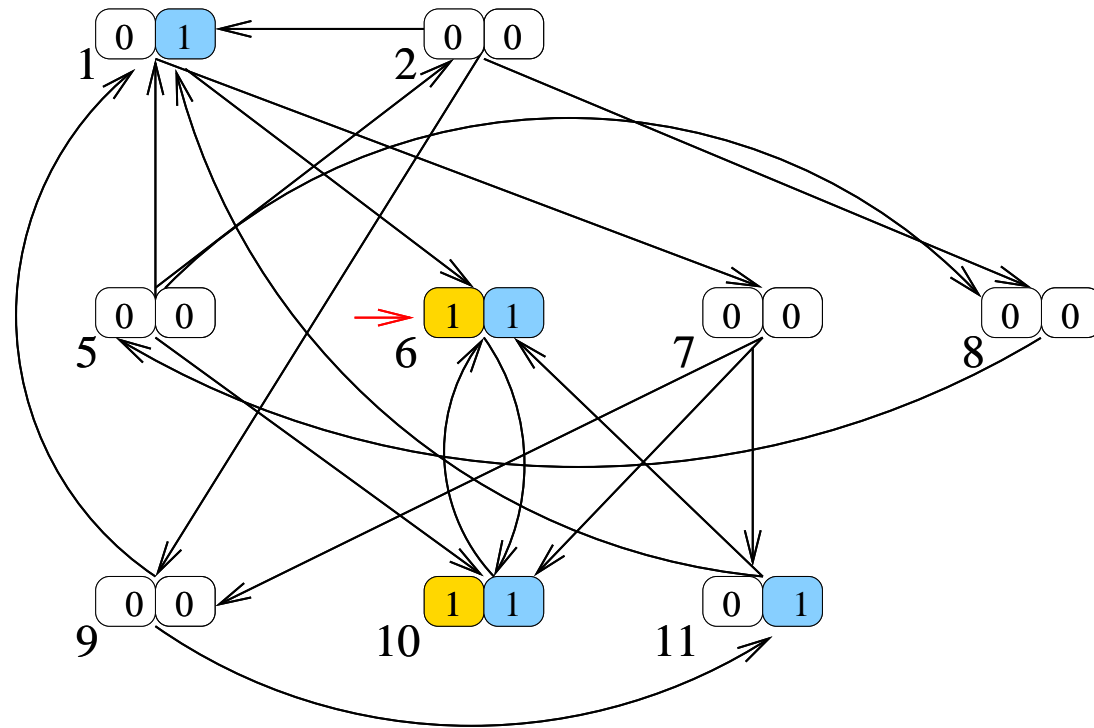
$$\begin{aligned} V_{00} &= V_{01} = \emptyset \\ V_{10} &= \{1, 2, 5, 6, 7, 8, 9, 10, 11\} \\ V_{11} &= \{4, 12\} \end{aligned}$$



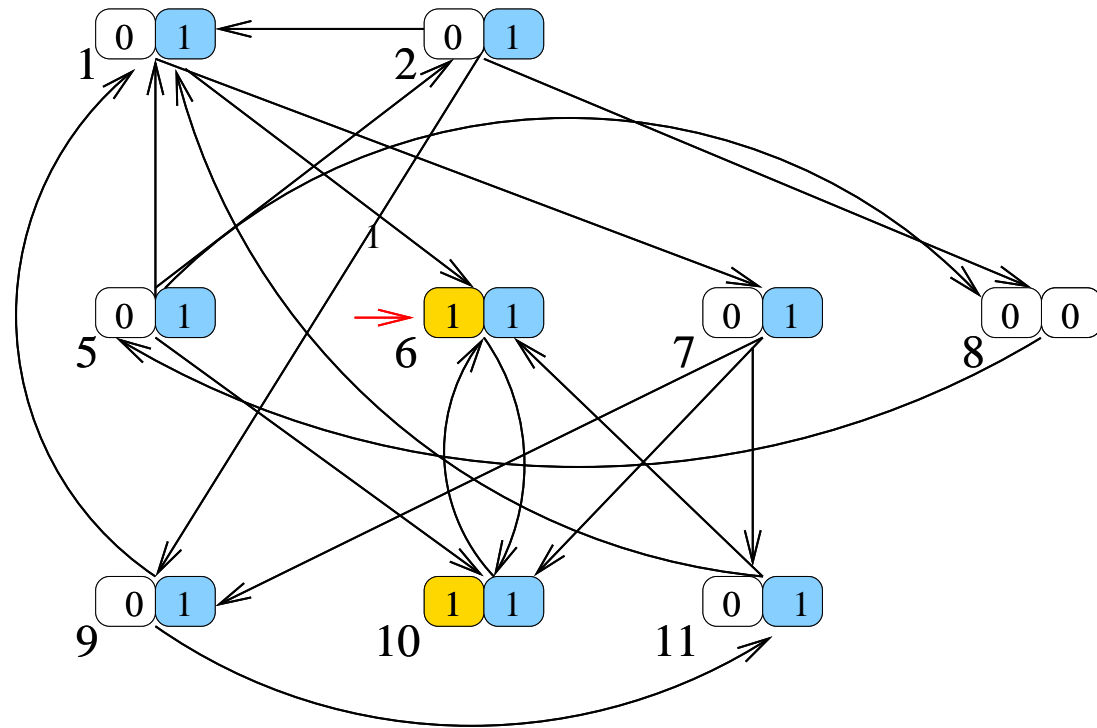
$S_1 = \{3\}$ $S_2 = \{4, 12\}$



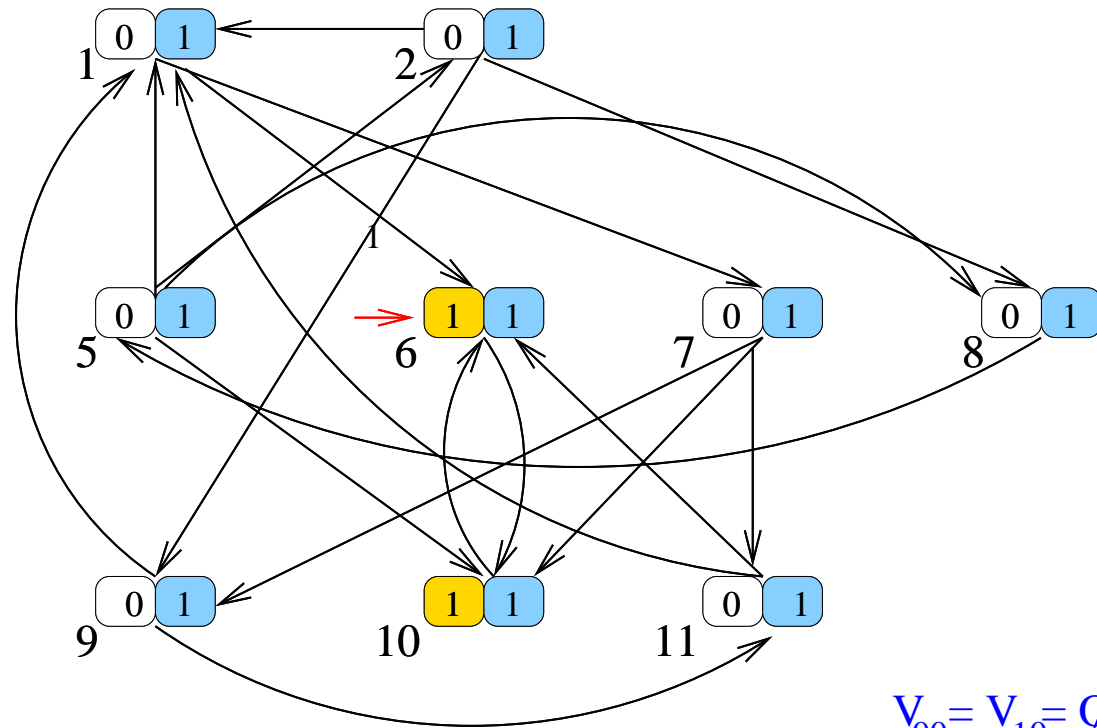
$$S_1 = \{3\} \quad S_2 = \{4, 12\}$$



$$S_1 = \{3\} \quad S_2 = \{4, 12\} \quad S_3 = \{6, 10\}$$



$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$

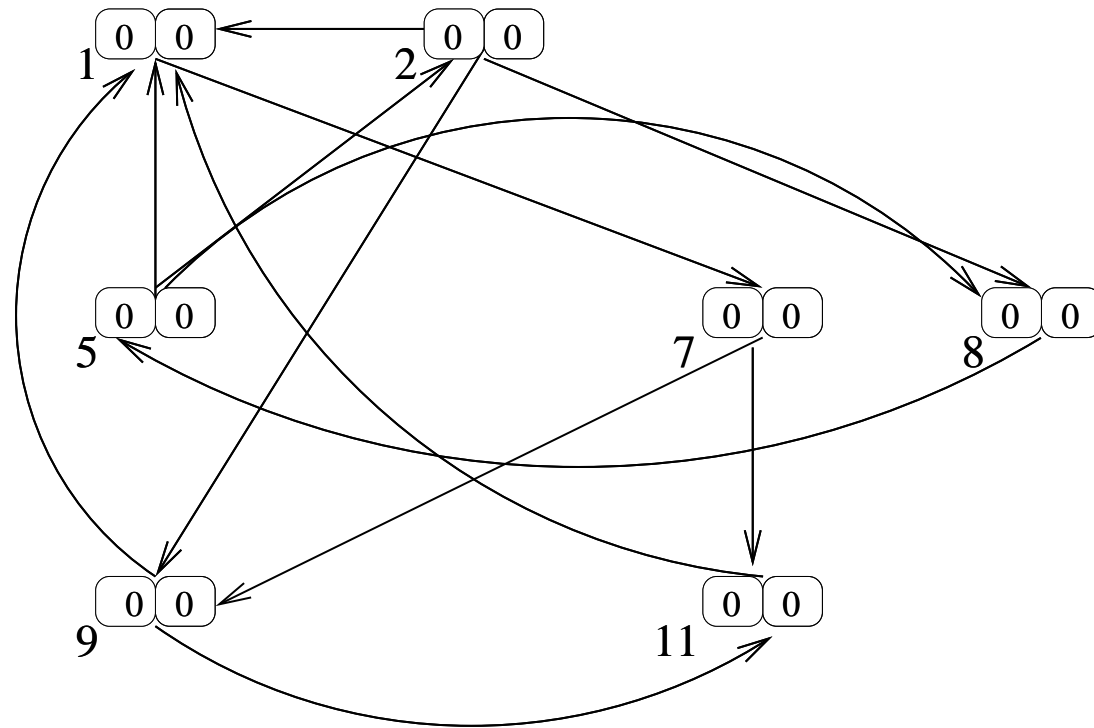


$$S_1 = \{3\} \quad S_2 = \{4, 12\} \quad S_3 = \{6, 10\}$$

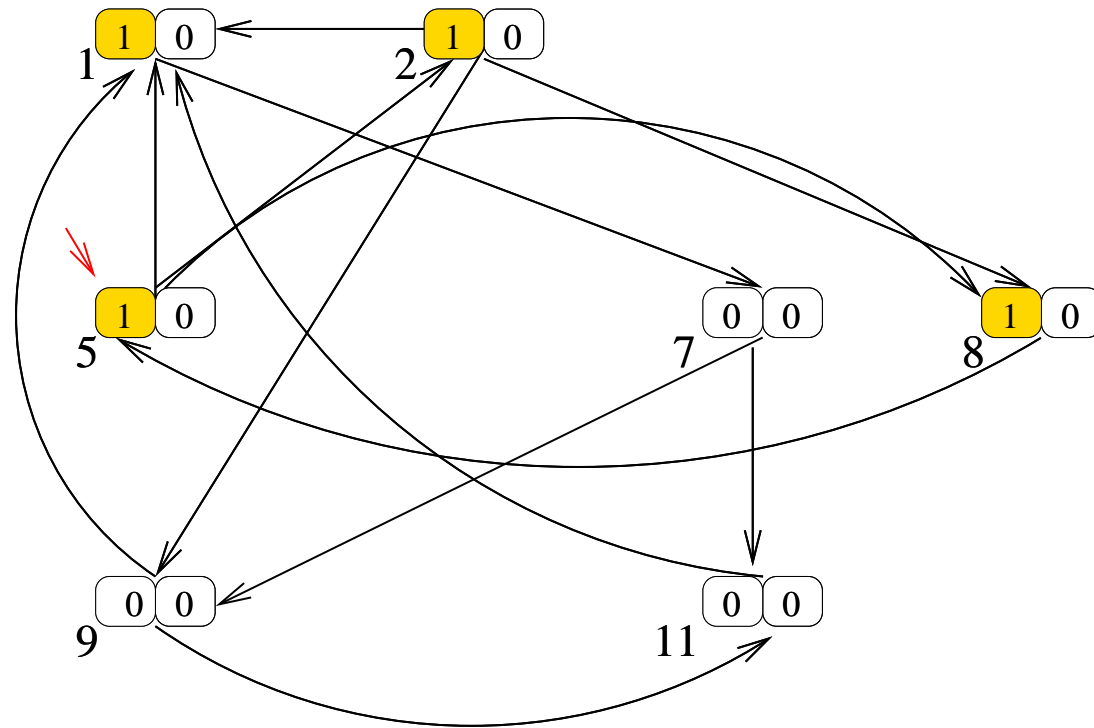
$$V_{00} = V_{10} = \emptyset$$

$$V_{01} = \{1, 2, 5, 7, 8, 9, 11\}$$

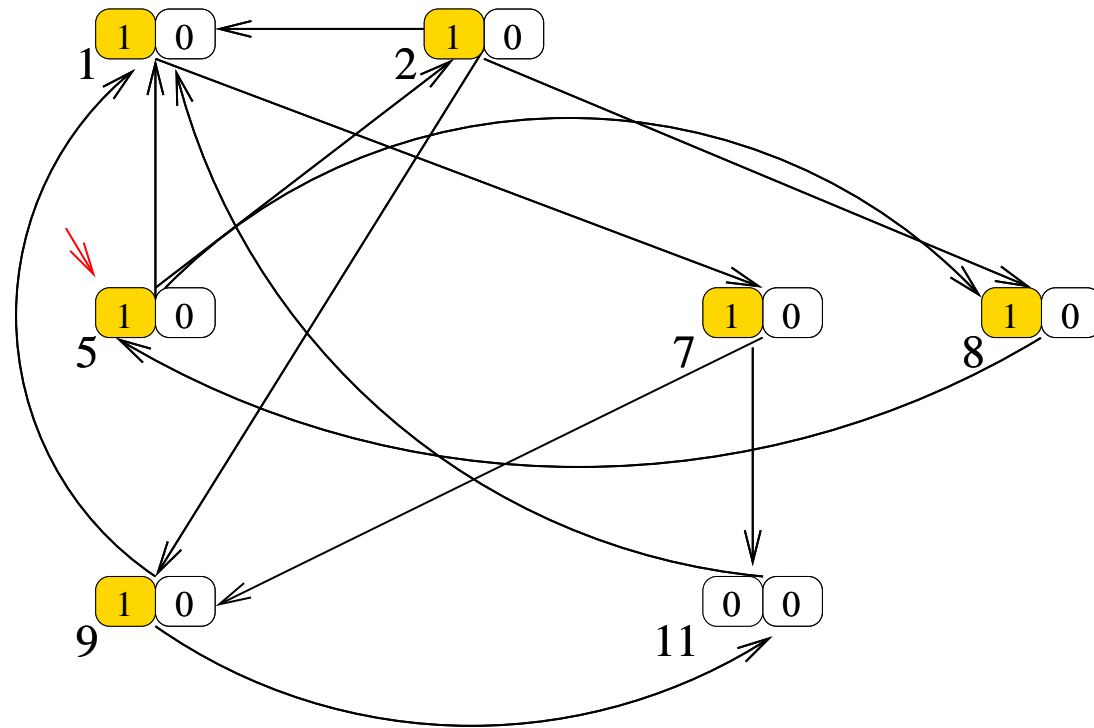
$$V_{11} = \{6, 10\}$$



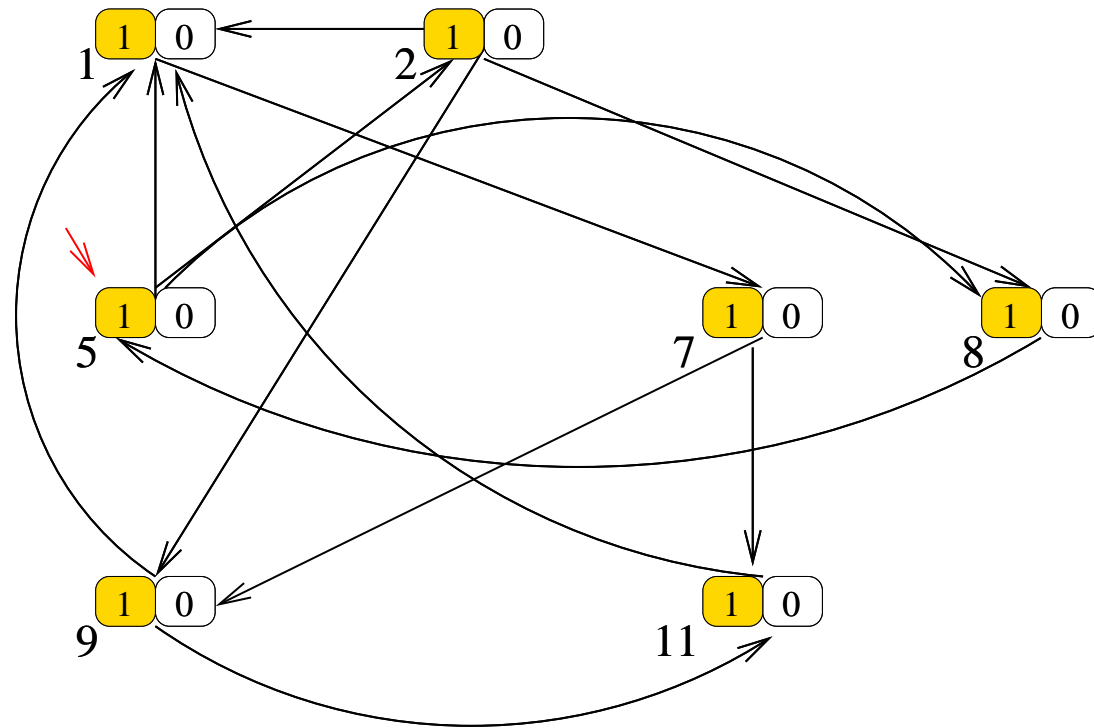
$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$



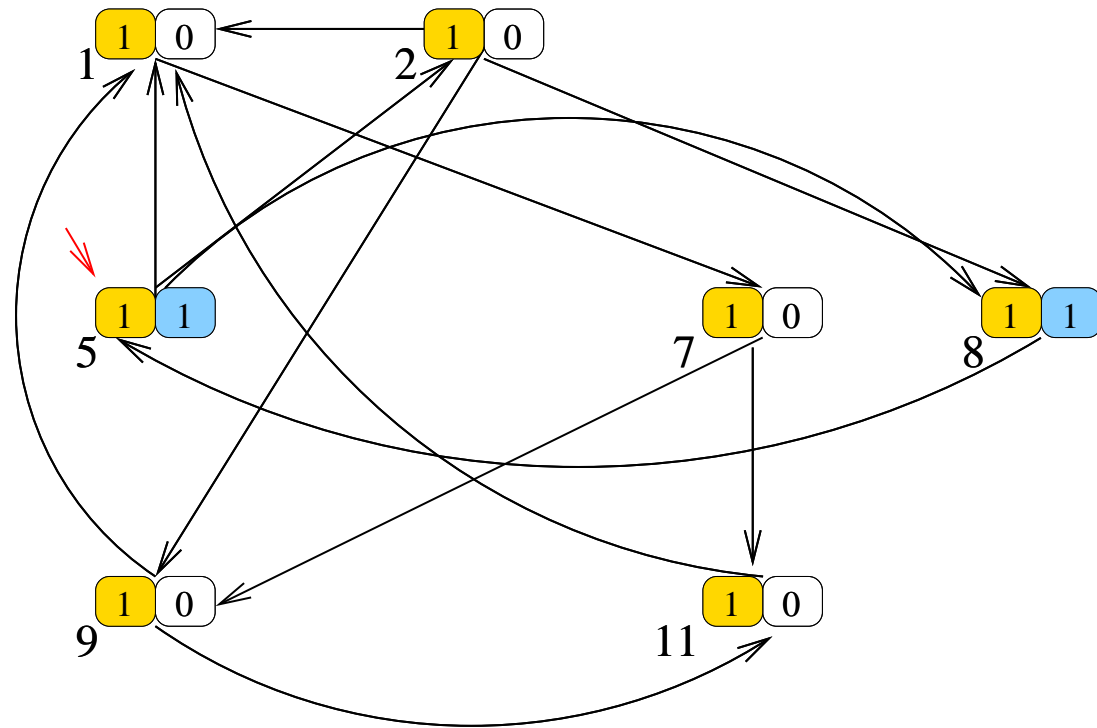
$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$



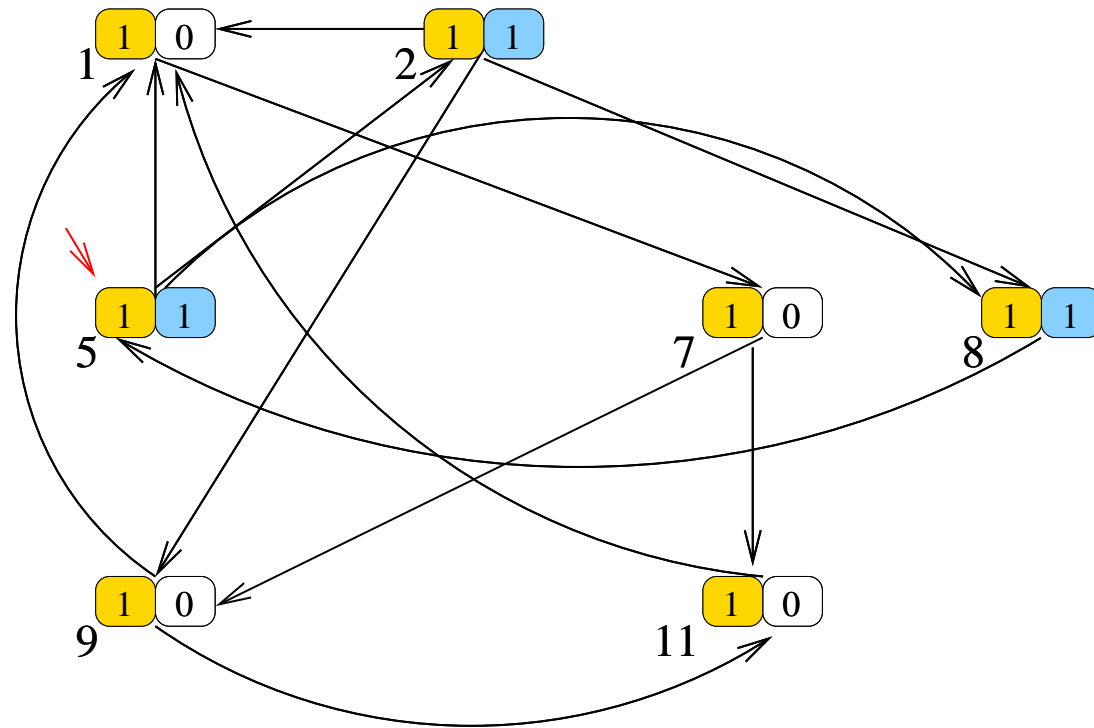
$$S_1 = \{3\} \quad S_2 = \{4, 12\} \quad S_3 = \{6, 10\}$$



$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$



$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$



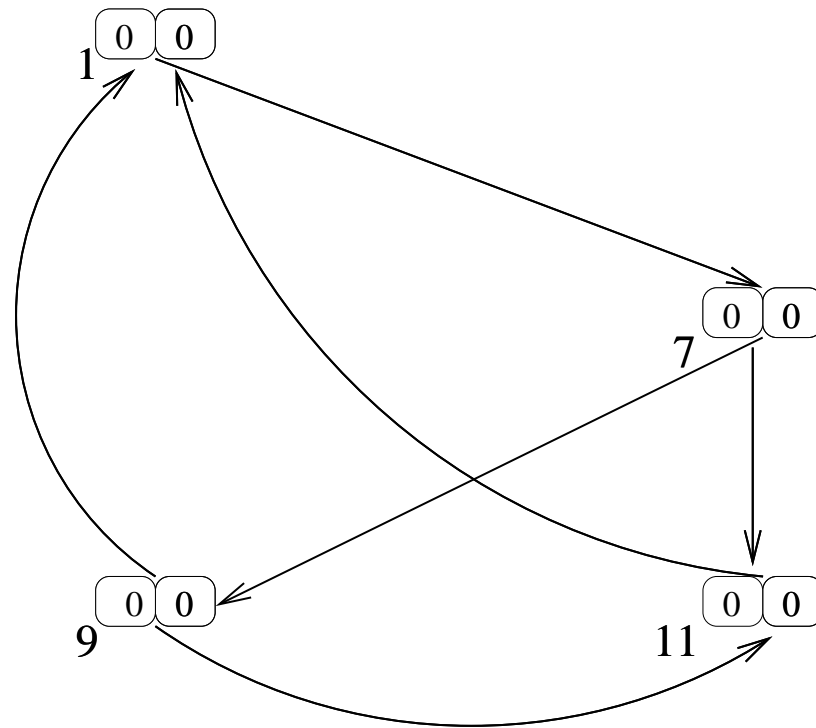
$$S_1 = \{3\} \quad S_2 = \{4, 12\} \quad S_3 = \{6, 10\}$$

$$S_4 = \{2, 5, 8\}$$

$$V_{00} = V_{01} = \emptyset$$

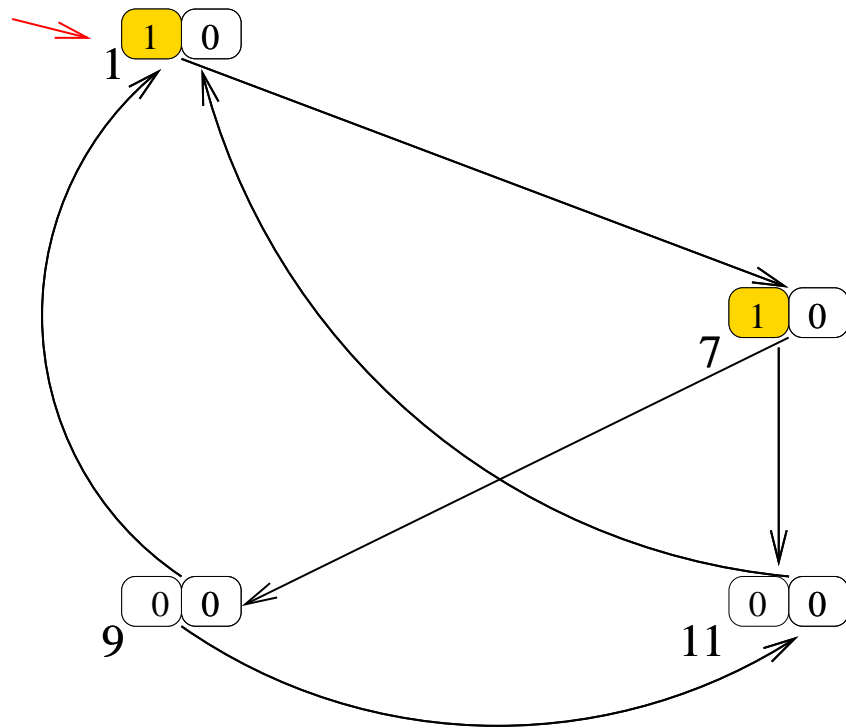
$$V_{10} = \{1, 7, 9, 11\}$$

$$V_{11} = \{2, 5, 8\}$$



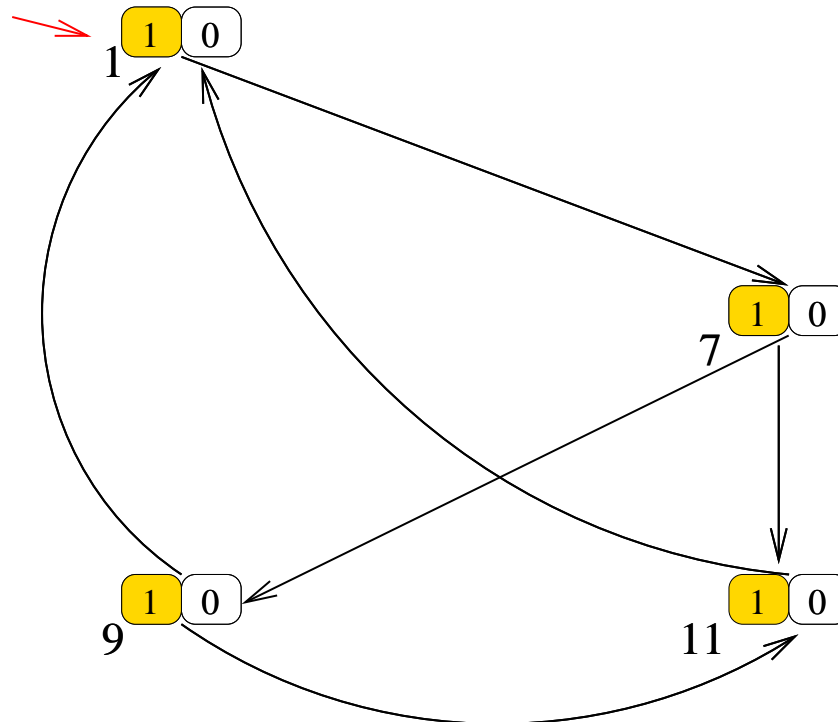
$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$

$S_4 = \{2, 5, 8\}$



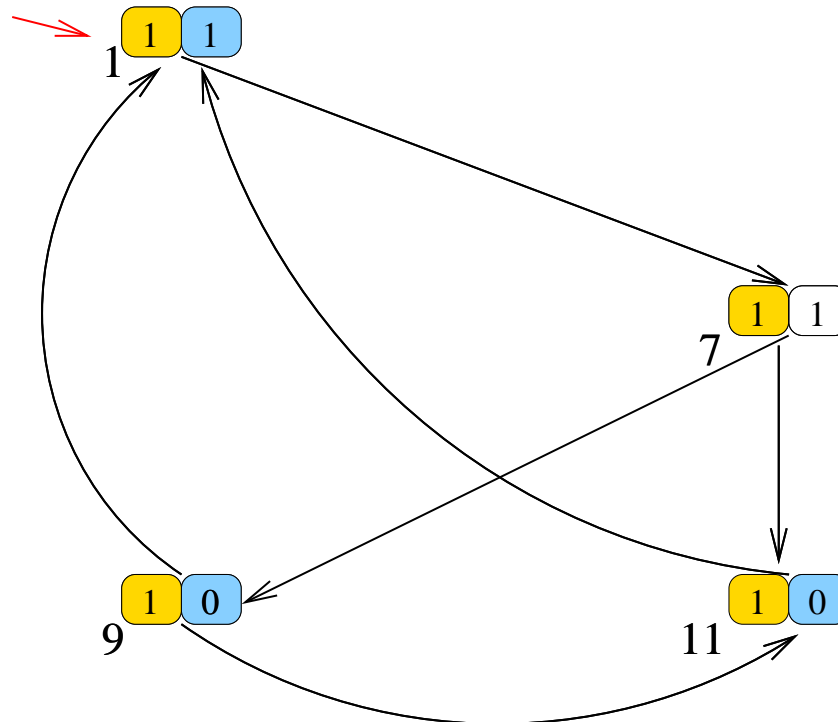
$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$

$S_4 = \{2, 5, 8\}$



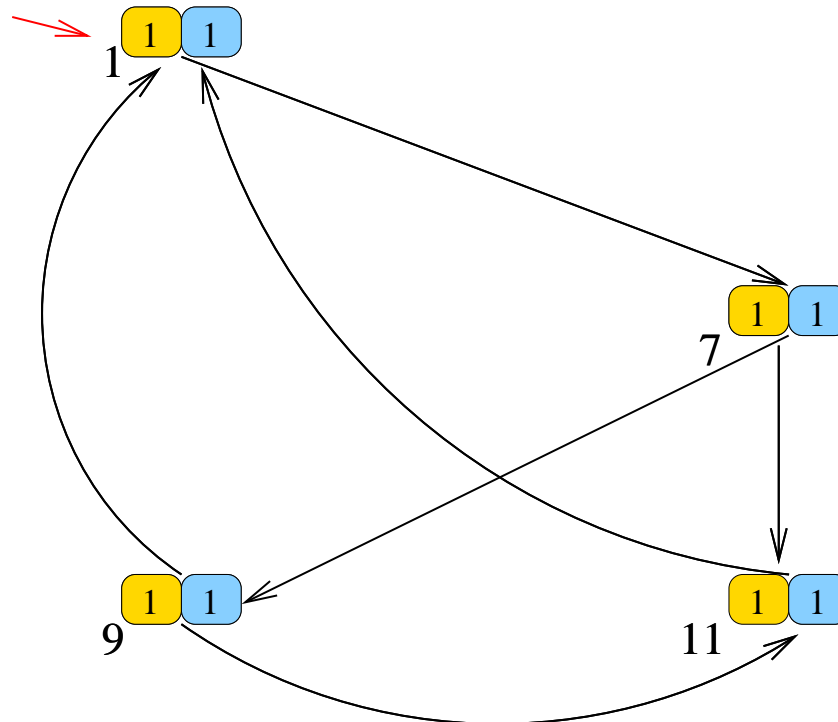
$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$

$S_4 = \{2, 5, 8\}$



$S_1 = \{3\}$ $S_2 = \{4, 12\}$ $S_3 = \{6, 10\}$

$S_4 = \{2, 5, 8\}$



$$S_1 = \{3\} \quad S_2 = \{4, 12\} \quad S_3 = \{6, 10\}$$

$$S_4 = \{2, 5, 8\} \quad S_5 = \{1, 7, 9, 11\}$$

$$V_{00} = V_{01} = V_{10} = \emptyset$$

$$V_{11} = \{1, 7, 9, 11\}$$

Algorytm wykorzystujący DFS

- podobna idea do algorytmu Leifmana (BFS na grafie skierowanym i jego transpozycji)
- transpozycja digrafu $G = (V, E)$: $G^T(V, E^T)$, gdzie $\{(u, v) : (v, u) \in E\}$
- kluczowa obserwacja sprowadza się do stwierdzenia, że digrafy G i G^T mają dokładnie te same składowe silnej spójności
- konstrukcja transpozycji grafu ma złożoność $O(|V| + |E|)$

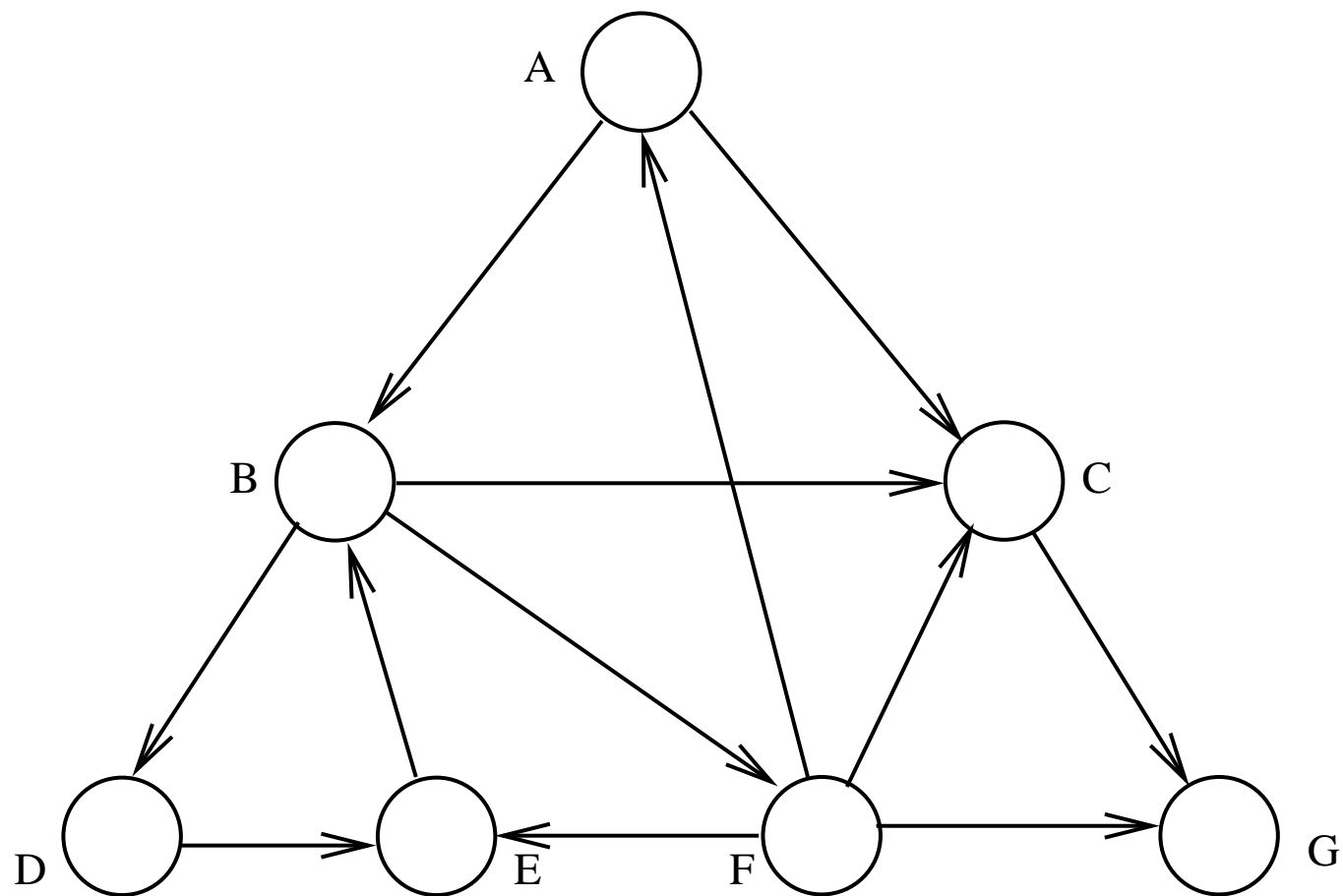
DFS-SILNE-SPÓJNE-SKŁADOWE(G)

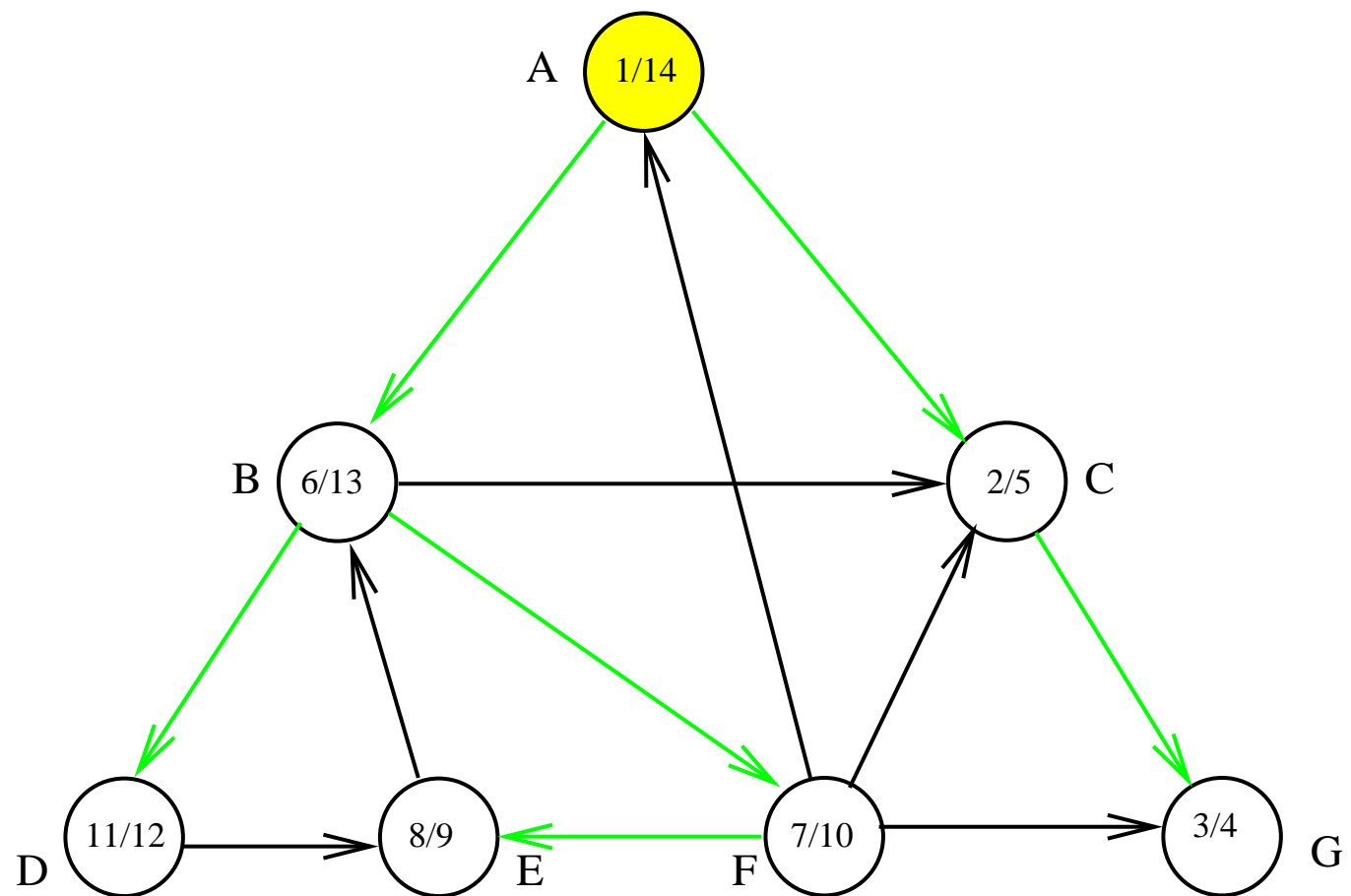
1. wykonaj DFS(G) w celu obliczenia czasu przetworzenia dla każdego wierzchołka grafu
2. oblicz transpozycję G^T digrafu G
3. wykonaj DFS(G^T) rozpoczynając budowę kolejnego drzewa w lesie przeszukiwania w głąb od wierzchołka o najwyższym czasie przetworzenia
4. wypisz wierzchołki z każdego drzewa w lesie przeszukiwania w głąb z kroku 3 jako oddzielną silnie spójną składową

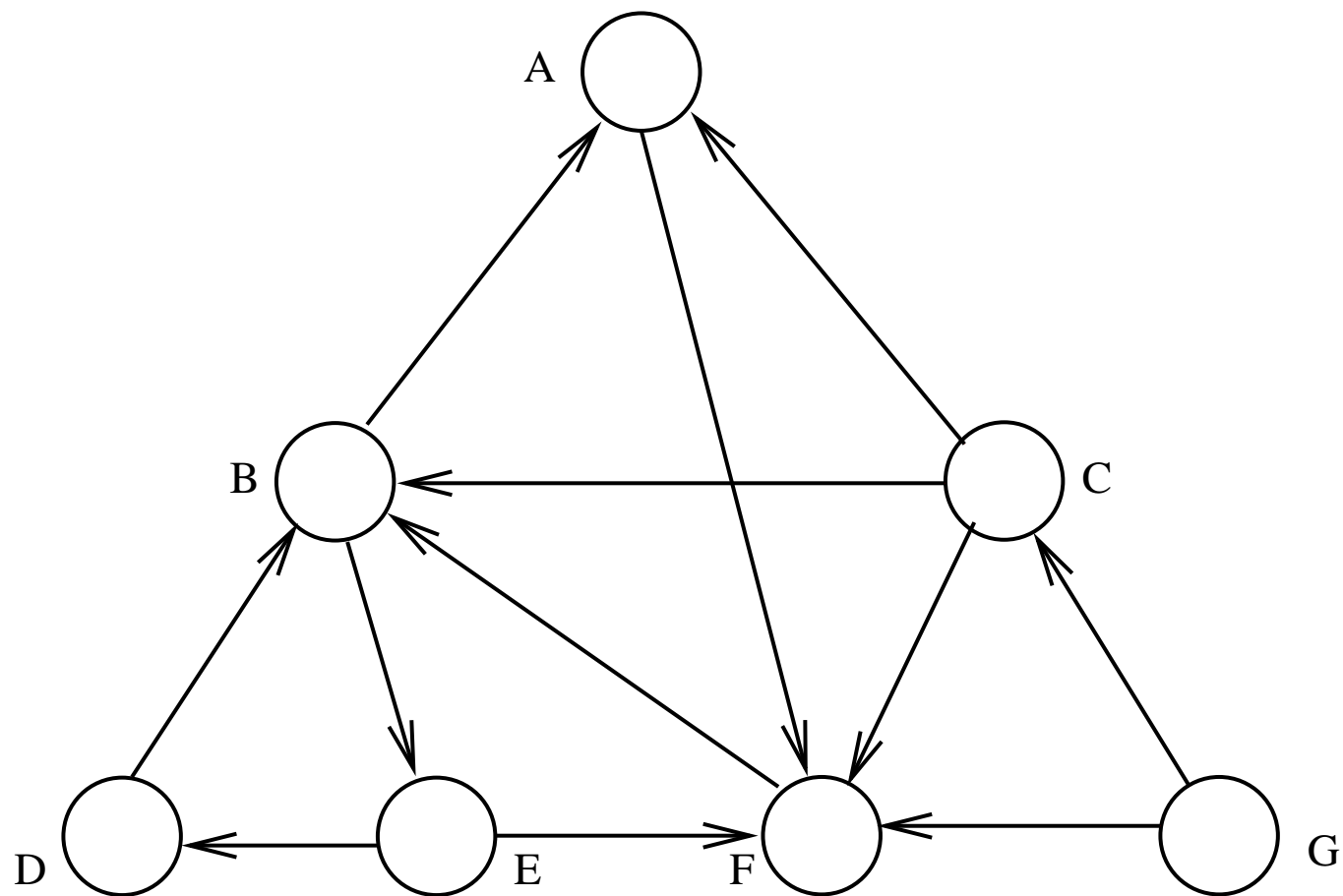
Twierdzenie . Algorytm DFS-SILNE-SPÓJNE-SKŁADOWE(G) poprawnie oblicza silnie spójne składowe grafu skierowanego G .

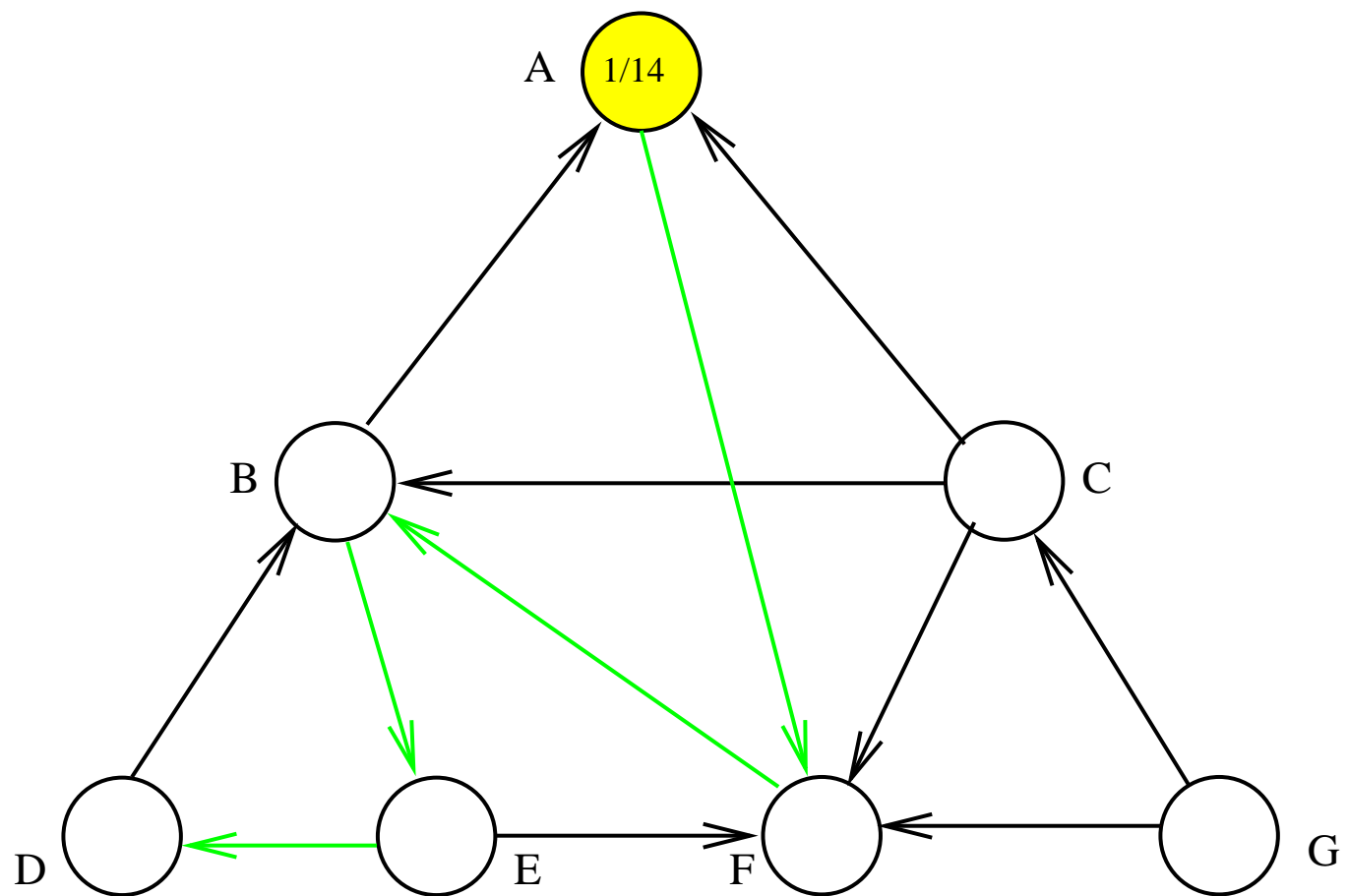


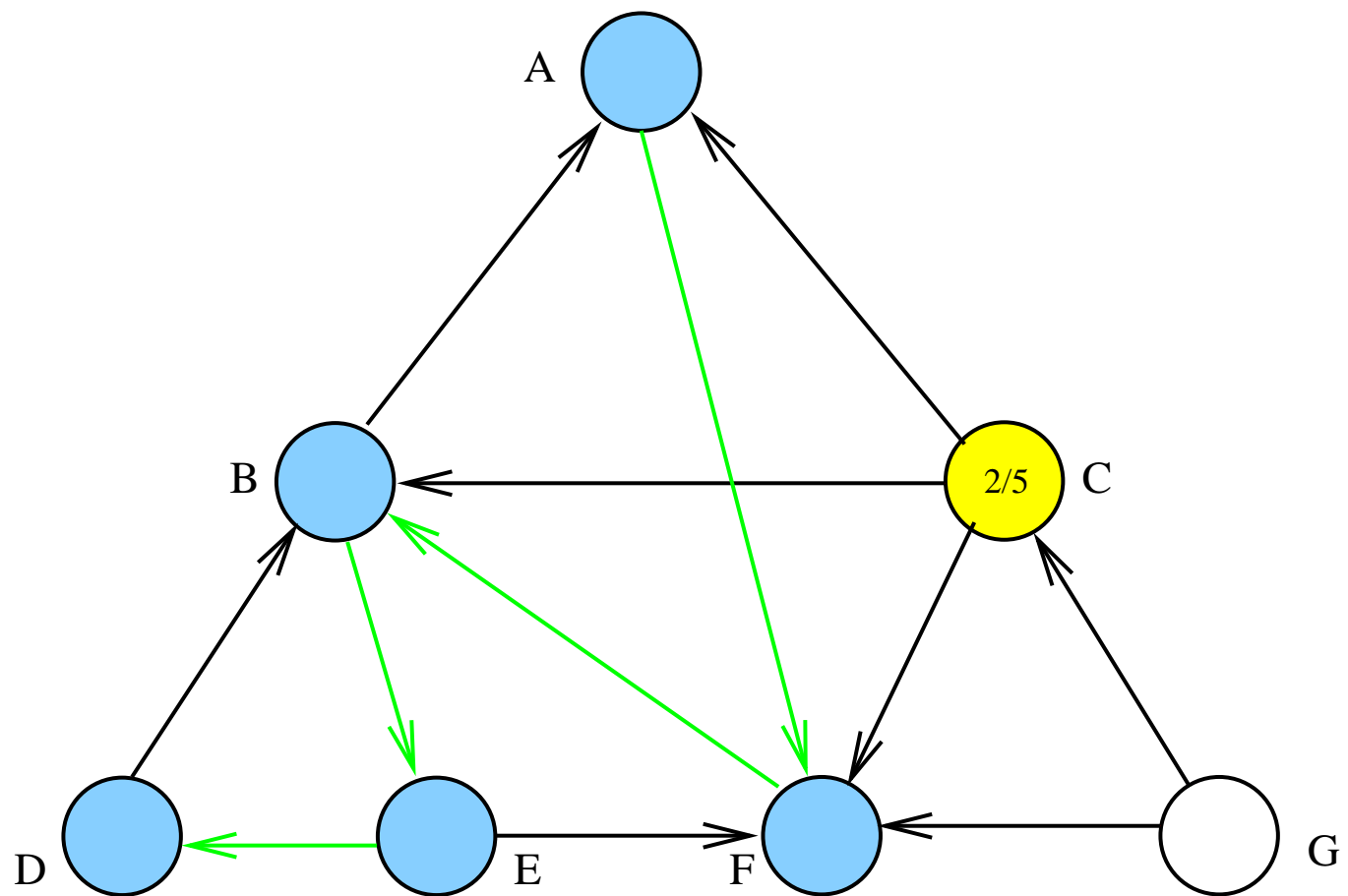
- szczegółowy dowód powyższego twierdzenia jest zawarty w paragrafie 23.8 książki Cormena, Leisersona i Rivesta
- ograniczymy się do zademonstrowania działania algorytmu na przykładzie znajdowania silnie spójnych składowych następującego grafu:

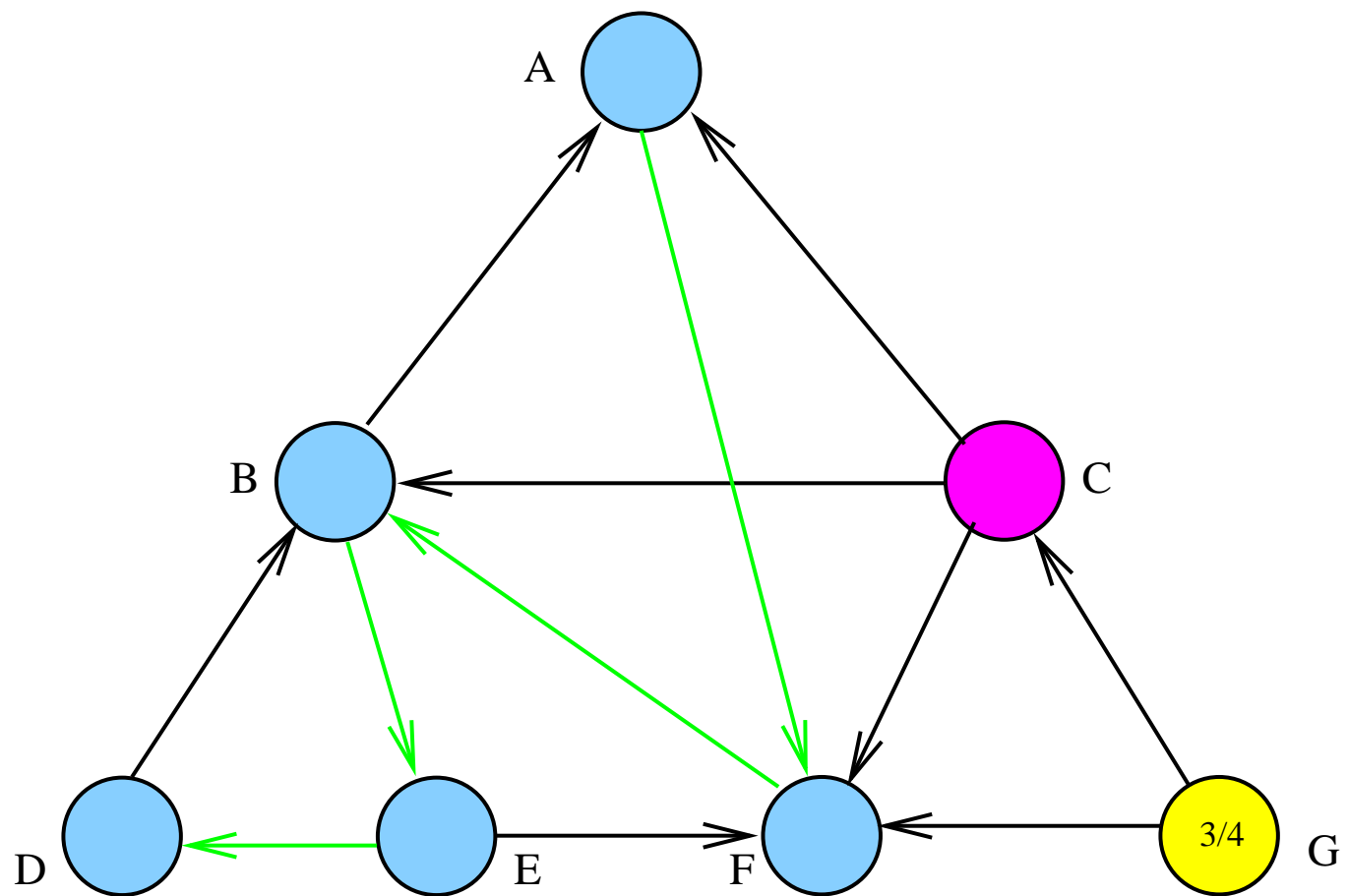


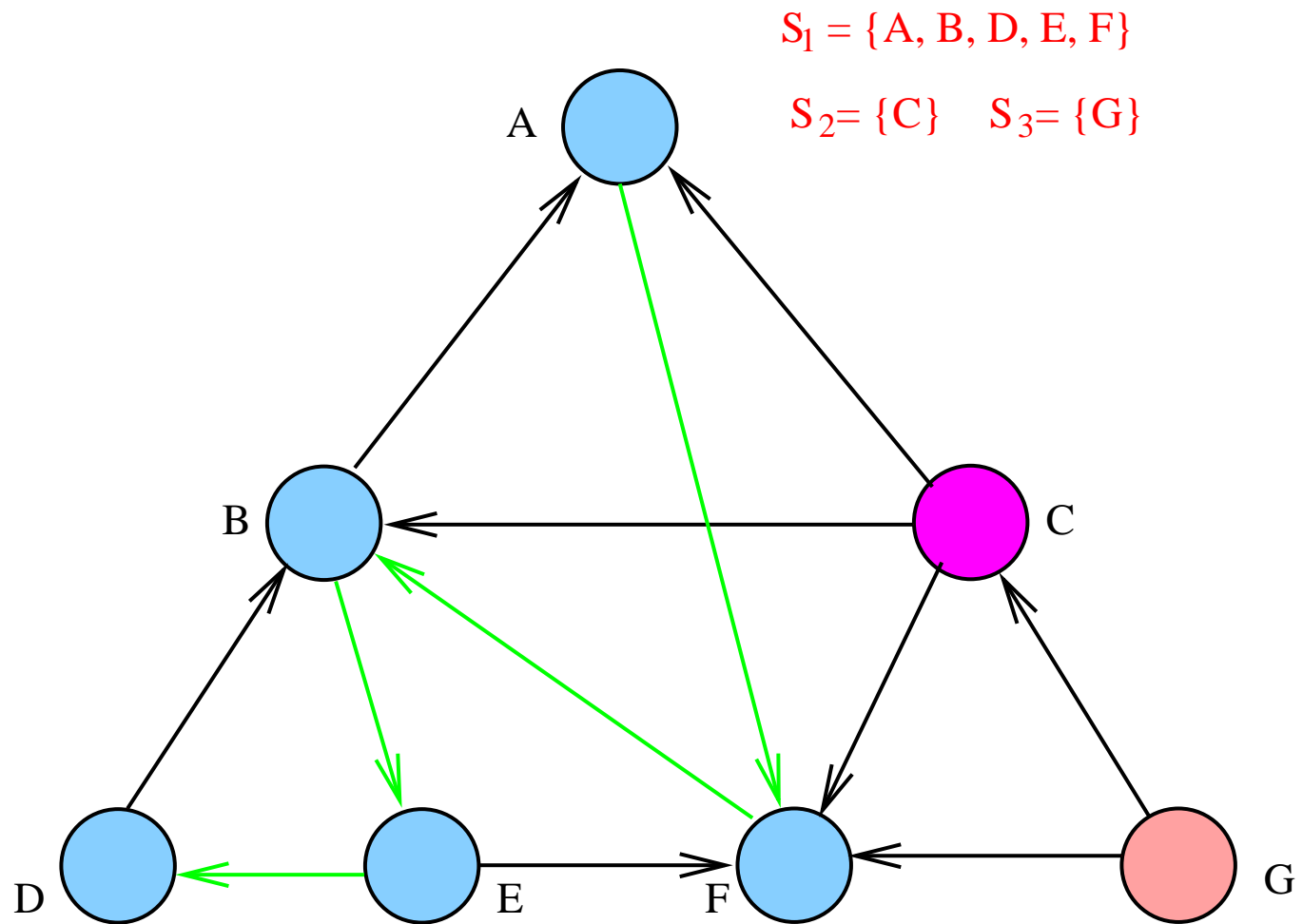


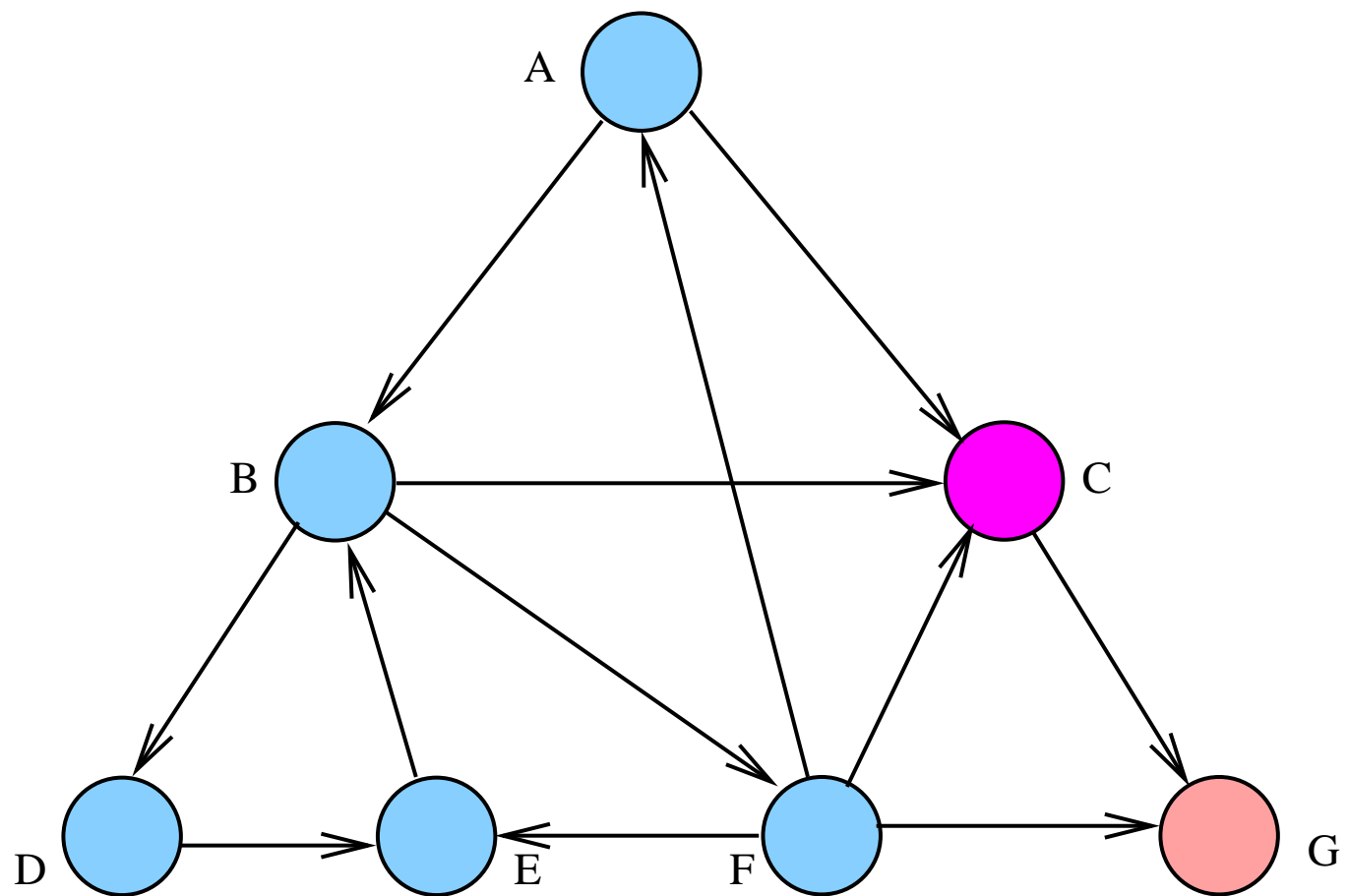










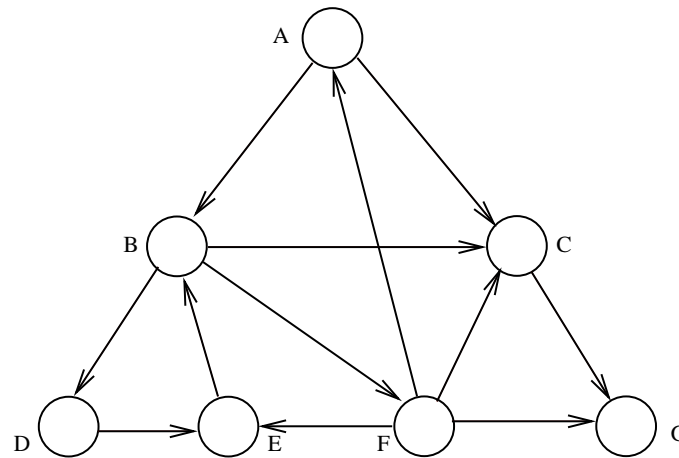


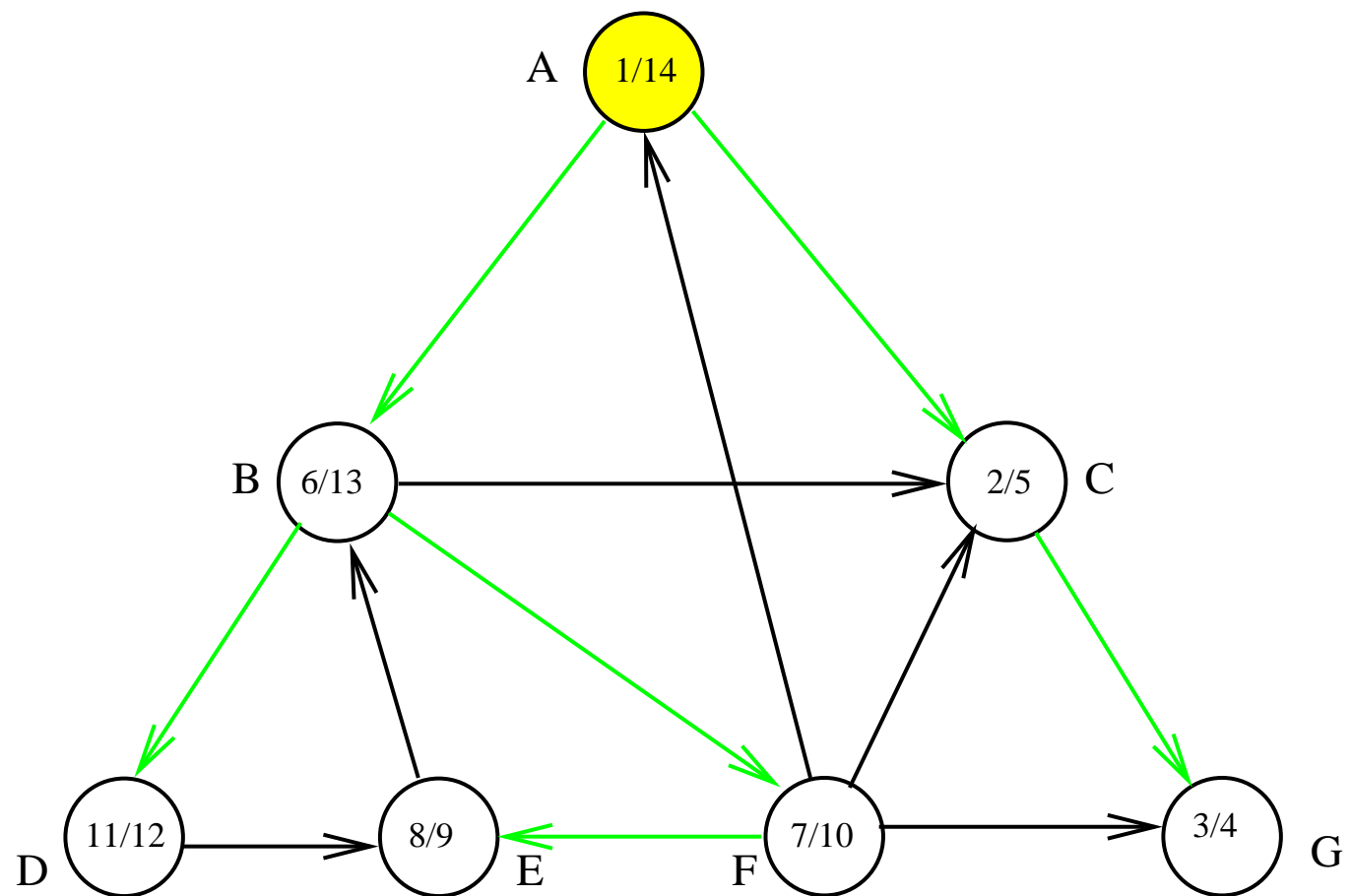
Przykład . Sortowanie semi-topologiczne

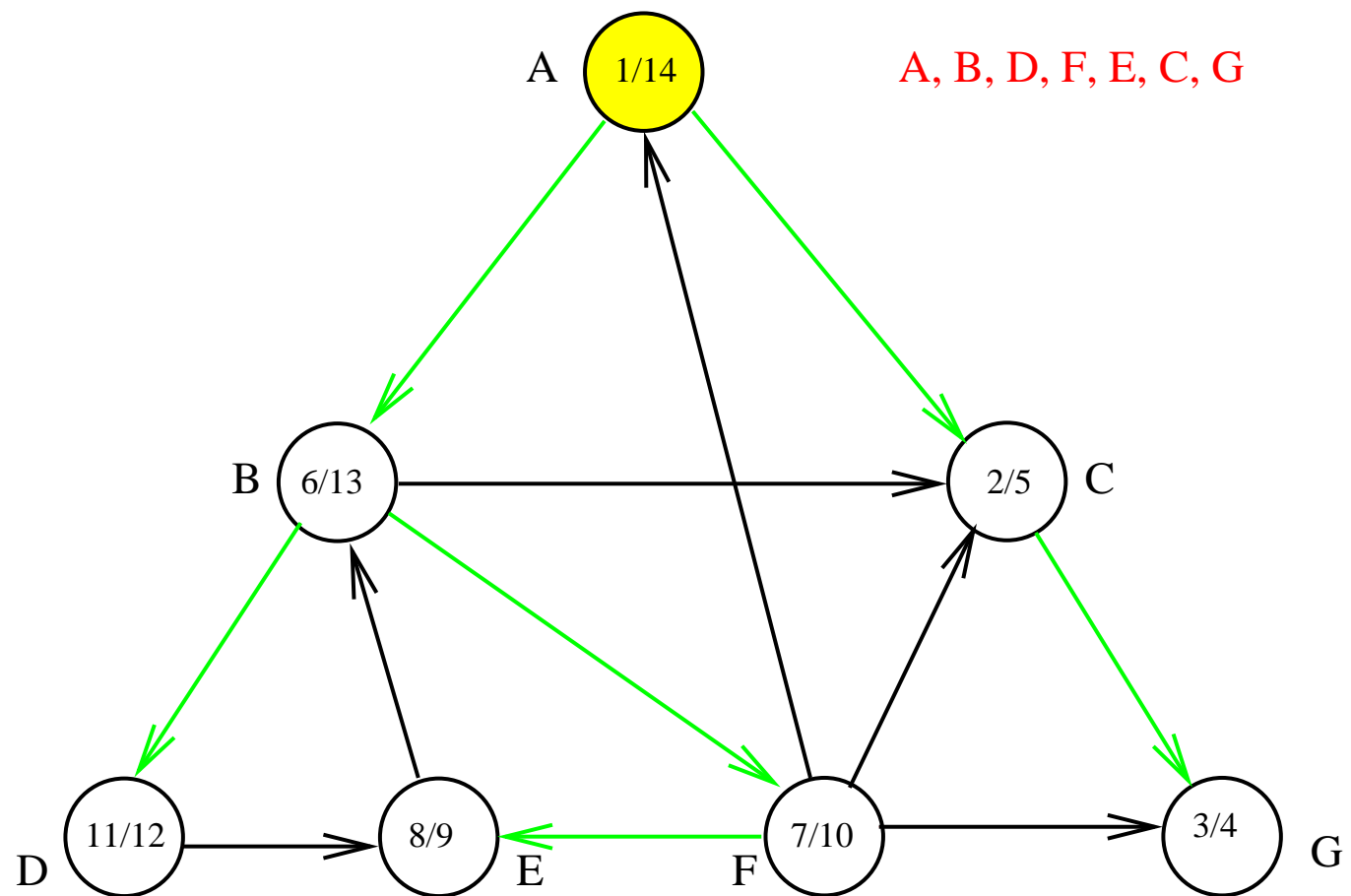
- $G = (V, E)$ - **dowolny** graf skierowany (digraf)
- sortowanie semi-topologiczne digrafu polega na takim liniowym uporządkowaniu jego wierzchołków, że jeżeli wierzchołek u występuje w tym uporządkowaniu przed wierzchołkiem v i wierzchołek u jest **osiągalny** z wierzchołka v to wierzchołek v jest **osiągalny** z wierzchołka u
- jeżeli digraf jest acykliczny (dag) to uporządkowanie semi-topologiczne jest topologiczne
- algorytm sortowania semi-topologicznego wierzchołków dowolnego digrafu jest identyczny z algorytmem sortowania topologicznego

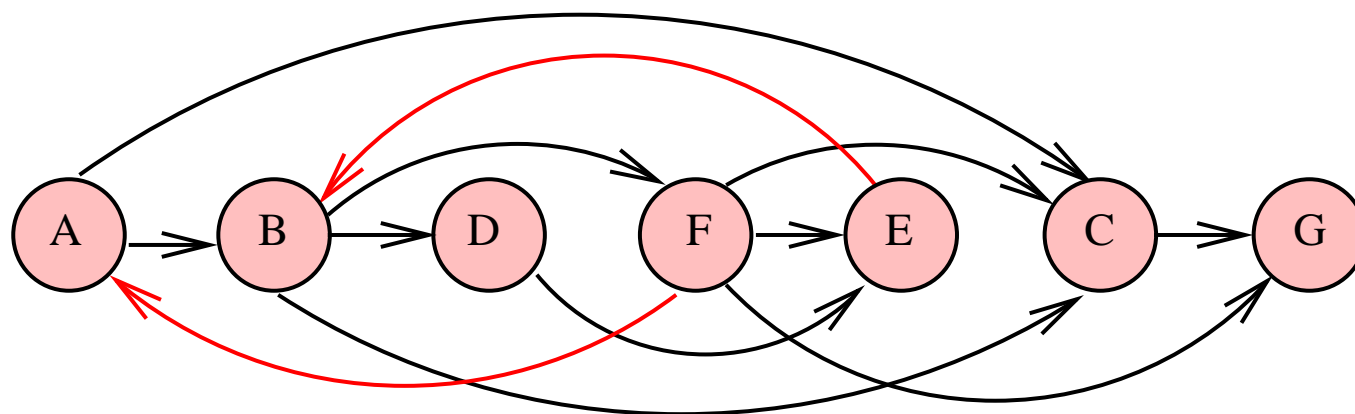
SORTOWANIE-SEMI-TOPOLOGICZNE(G)

- wykonaj DFS(G) w celu obliczenia czasów przetworzenia poszczególnych wierzchołków
- wstaw wierzchołek v na początek listy, kiedy tylko zostanie on przetworzony









5. Najkrótsze ścieżki

5.1 Odległości w grafach: definicje i własności

Definicja (Długość ścieżki). Długością ścieżki nazywamy liczbę krawędzi występujących w tej ścieżce. Bardziej formalnie, jeżeli

$$W = v_0 e_1 v_1 e_2, \dots, e_k v_k$$

jest ścieżką, to jej długością k (bo występują w niej kolejno krawędzie (łuki) e_1, e_2, \dots, e_k).

Definicja (Odległość wierzchołków). Jeżeli dwa wierzchołki v i u należą do tej samej składowej spójności grafu $G = (V, E)$ to ich odległość $\rho(v, u) = \rho_G(v, u)$ jest długością najkrótszej (v, u) -ścieżki w G , w przeciwnym przypadku (v i u należą do różnych składowych) $\rho(v, u) = \infty$.

- jeżeli krawędziom grafu zostały przyporządkowane pewne liczby (*wagi krawędzi*), to graf z taką dodatkowo określoną na zbiorze krawędzi funkcją nazywać będziemy *grafem z wagami*
- długość ścieżki w takim grafie to suma wag jej krawędzi
- graf bez wag możemy traktować jako graf z wagami jednostkowymi

Definicja (Średnica grafu). Średnicą grafu $G = (V, E)$, oznaczaną $\text{diam}(G)$, nazywamy największą odległość między wierzchołkami grafu, to jest

$$\text{diam}(G) = \max\{\rho_G(v, u) : v, u \in V(G)\}.$$

Definicja (Promień grafu). Promieniem grafu $G = (V, E)$, oznaczanym $r(G)$, nazywamy wielkość

$$r(G) = \min_{v \in V(G)} \max\{\rho_G(v, u) : u \in V(G)\}.$$

Wierzchołki dla których osiągnięte jest powyższe minimum, to znaczy wierzchołki $v \in V(G)$ takie, że

$$\max\{\rho_G(v, u) : u \in V(G)\} = r(G),$$

nazywamy *wierzchołkami centralnymi*, a zbiór wierzchołków centralnych nazywamy *centrum grafu*.

1. Najkrótsza ścieżka między dwoma wierzchołkami (odległość wierzchołków).
2. Najkrótsze ścieżki z dowolnego wierzchołka do pozostałych wierzchołków.
3. Najkrótsze ścieżki między wszystkimi parami wierzchołków (promień, średnica i centrum grafu).

5.1 Najkrótsze ścieżki: wagi jednostkowe

Opis algorytmu znajdowania najkrótszej ścieżki między dwoma ustalonymi wierzchołkami (przy pomocy **BFS**).

1. Etykietujemy wierzchołek s cechą 0 ; $i := 0$.
2. Znajdujemy wszystkich niezaetykietowanych jeszcze sąsiadów (wszystkie następni w przypadku grafu skierowanego) wierzchołków zaetykietowanych cechą i . Jeżeli takich wierzchołków nie ma, to STOP.
3. Etykietujemy wszystkie wierzchołki znalezione w kroku 2 cechą $i + 1$.
4. Jeżeli wierzchołek t został zaetykietowany, to STOP. W przeciwnym razie zmieniamy $i := i + 1$ i wracamy do kroku 2.

- bardziej precyzyjnie procedura ta przedstawiona jest w postaci pseudokodu jako algorytm DISTBFS
- w algorytmie tym d jest tablicą wyznaczonych już odległości.
- wewnątrz pętli **repeat** P jest zbiorem wierzchołków o odległości $k - 1$ od s ,
- N jest aktualnie wyznaczanym zbiorem wierzchołków o odległości k
- $\Gamma(v)$ to zbiór sąsiadów wierzchołka v
- działanie programu kończy się, gdy znajdziemy odległość s od t (warunek $w = t$) lub znajdziemy odległości s od wszystkich wierzchołków w składowej zawierającej s (warunek $N = \emptyset$).

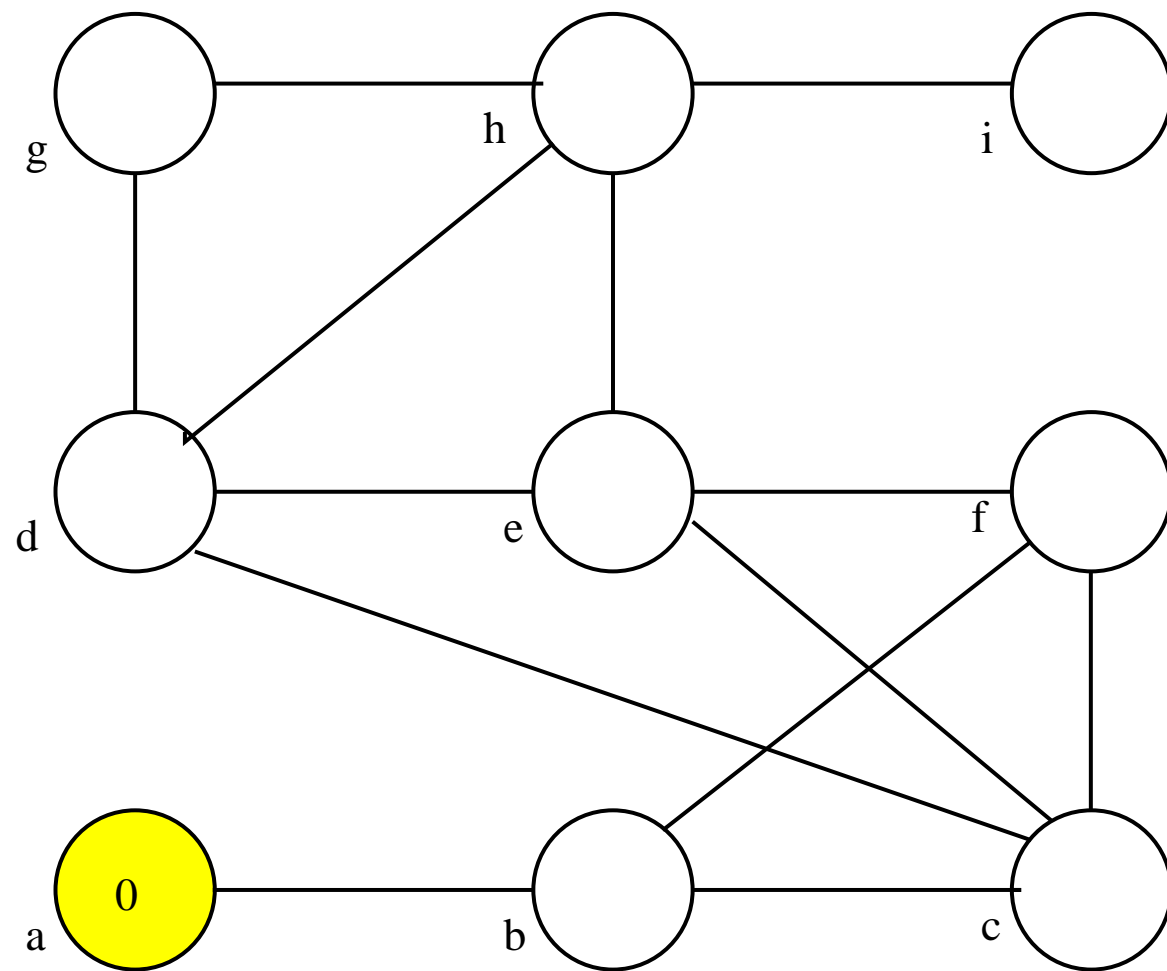
Algorithm 0.0.1: DISTBFS(G, s, t)

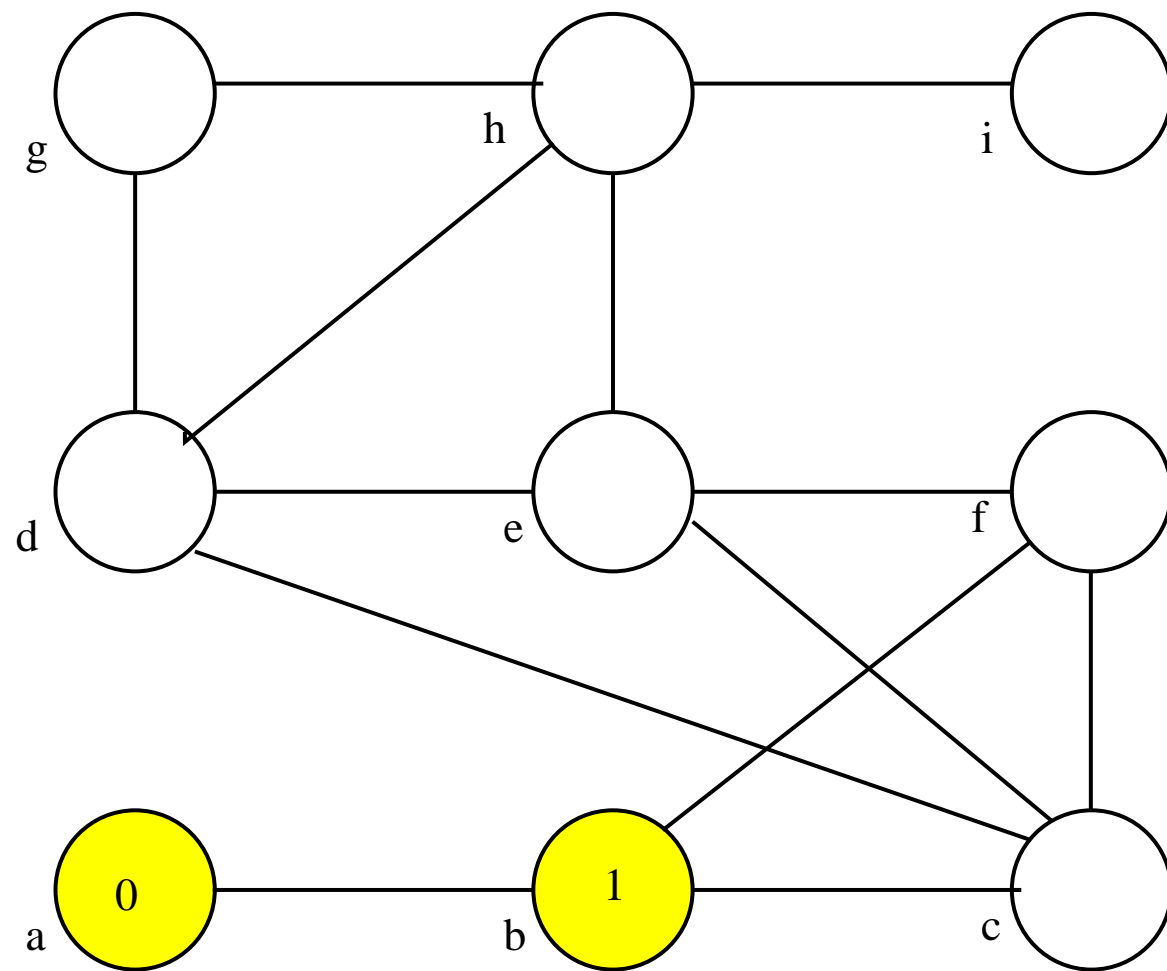
```
for each  $v \in V(G)$ 
  do  $d[v] \leftarrow \infty$ 
 $d[s] \leftarrow 0$ ;  $P \leftarrow \{s\}$ ;  $k \leftarrow 0$ 
repeat
   $P \leftarrow N$ ;  $N \leftarrow \emptyset$ ;  $k \leftarrow k + 1$ 
  for each  $v \in P$ 
    do for each  $w \in \Gamma(v)$ 
      do if  $d[w] = \infty$ 
        then  $\begin{cases} d[w] \leftarrow k \\ N \leftarrow N \cup \{w\} \\ \text{if } w = t \text{ then break} \end{cases}$ 
  until  $N = \emptyset$ 
return ( $d[t]$ )
```

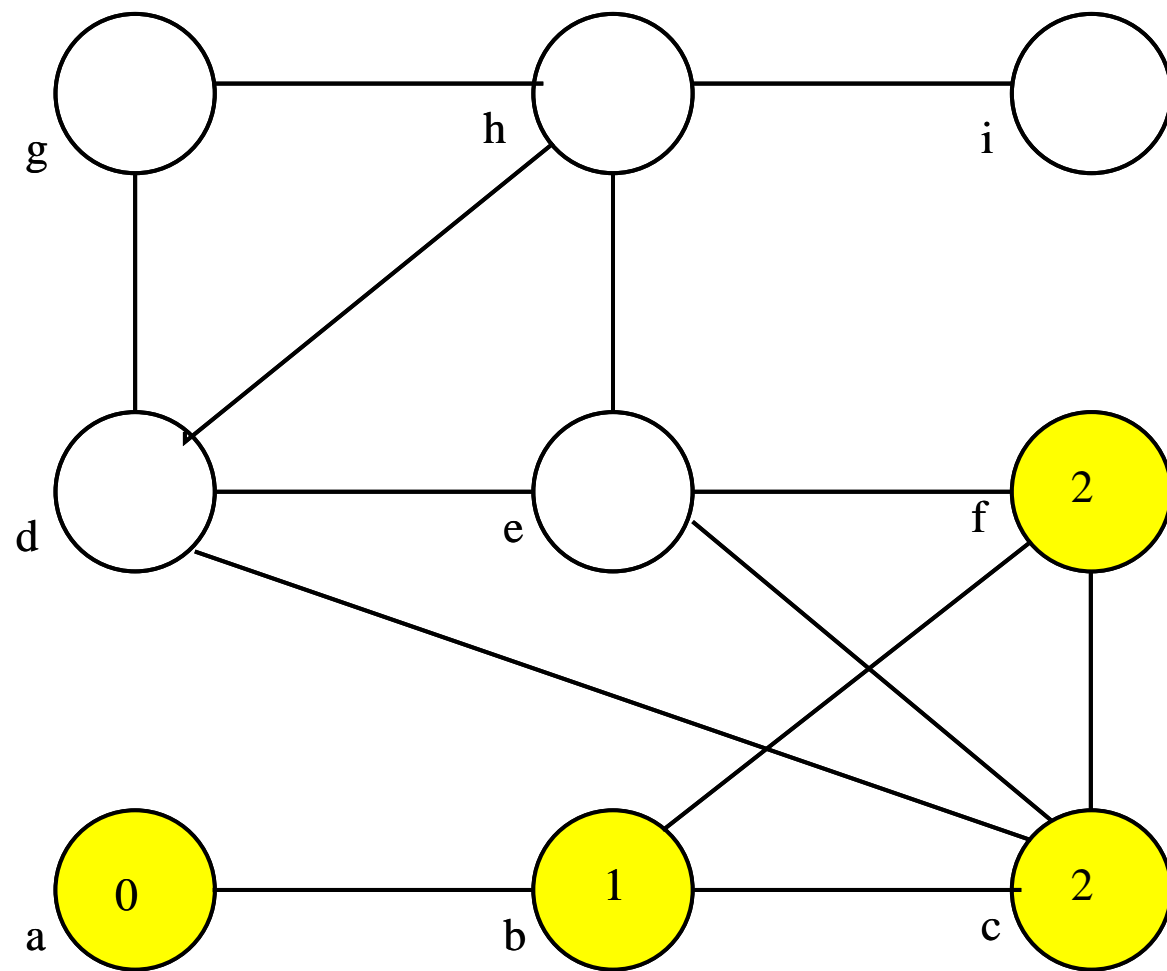
- Prosta modyfikacja tego algorytmu – usunięcie instrukcji **if $w = t$ then break** spowoduje obliczenie, w tablicy d , odległości wierzchołka s od wszystkich wierzchołków grafu (to znaczy, po wyjściu z procedury będzie zachodziła dla wszystkich $v \in V(G)$ równość $d[v] = \rho(s, v)$).

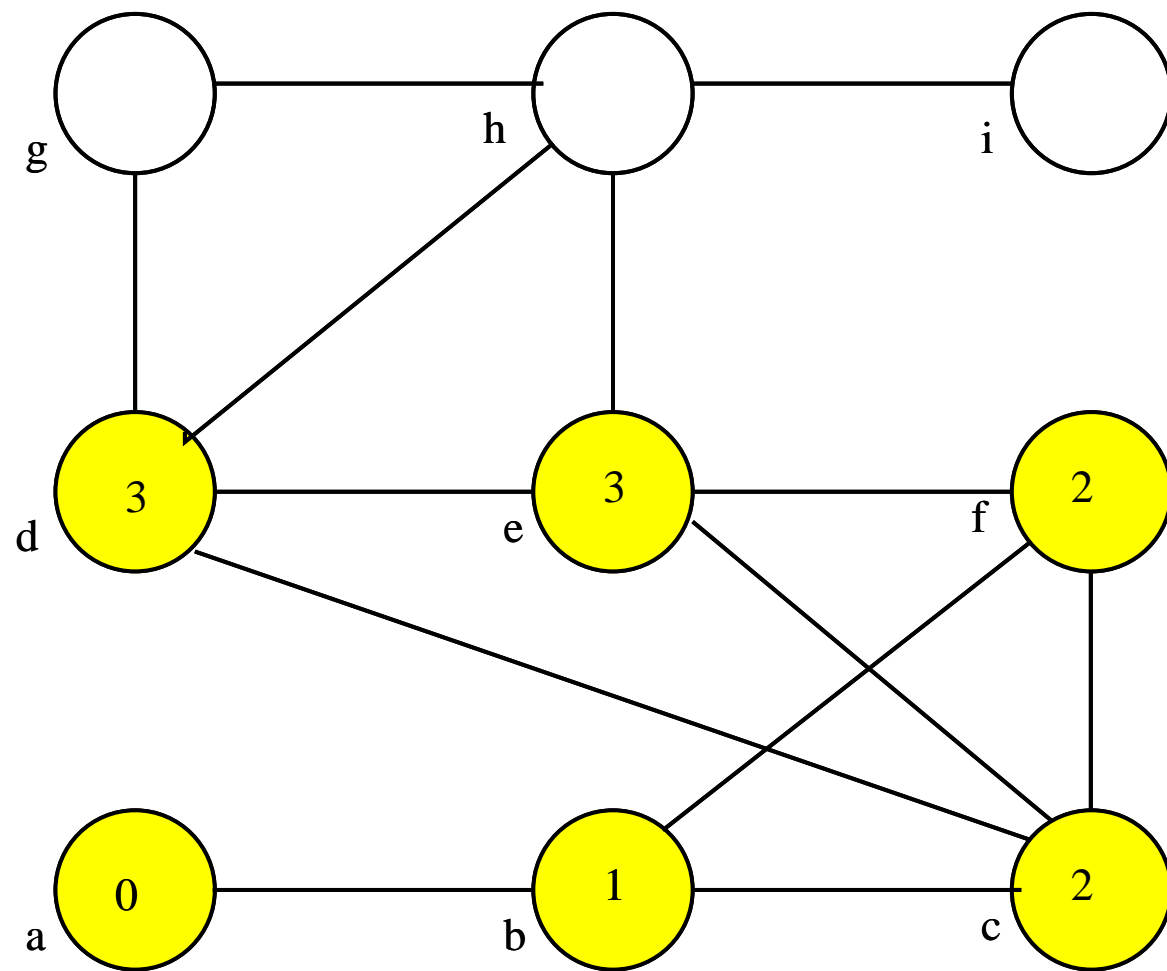
Przykład . Wyznacz odległość $\rho(a, i)$ w grafie zadanego macierzą przyległości

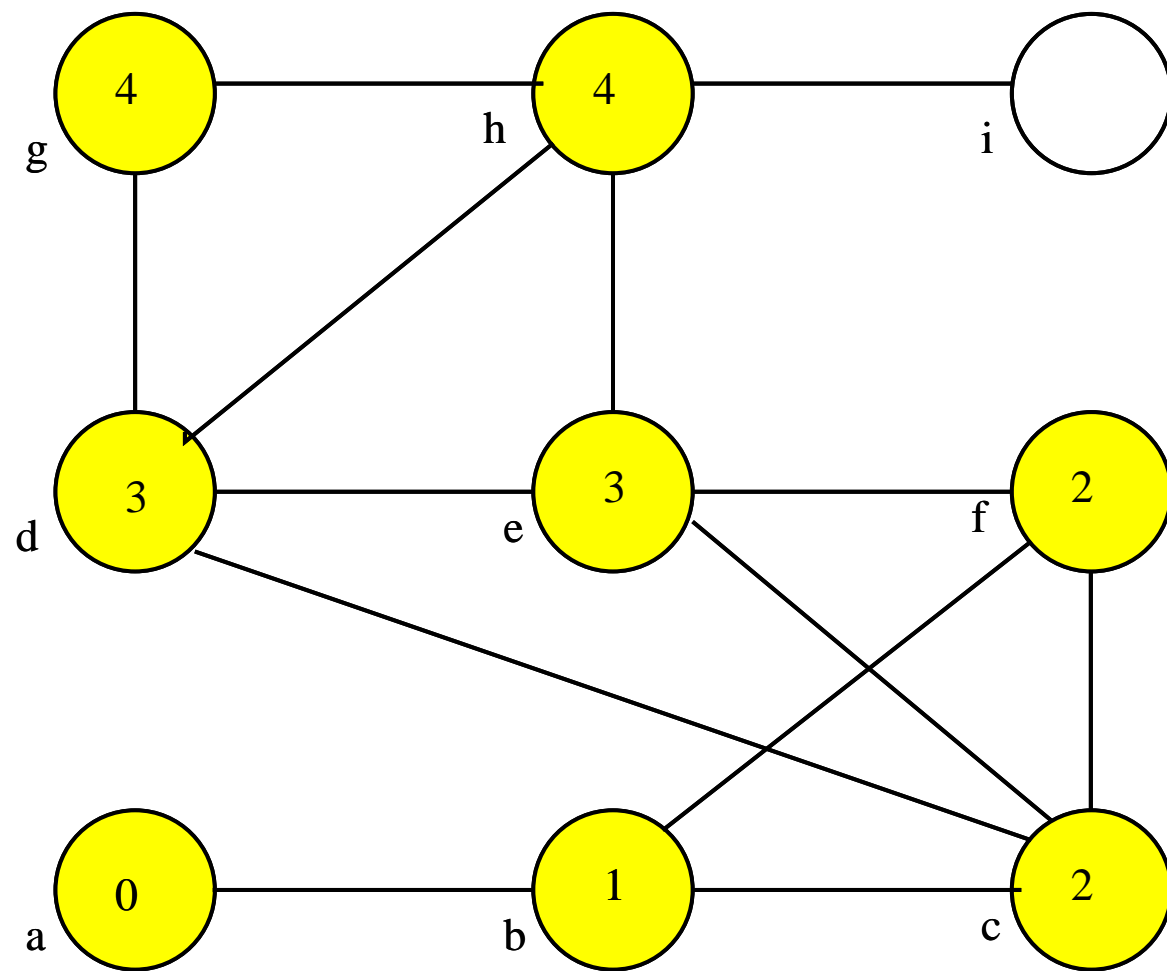
$$A(G) = \begin{matrix} & \begin{matrix} a & b & c & d & e & f & g & h & i \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{matrix} .$$

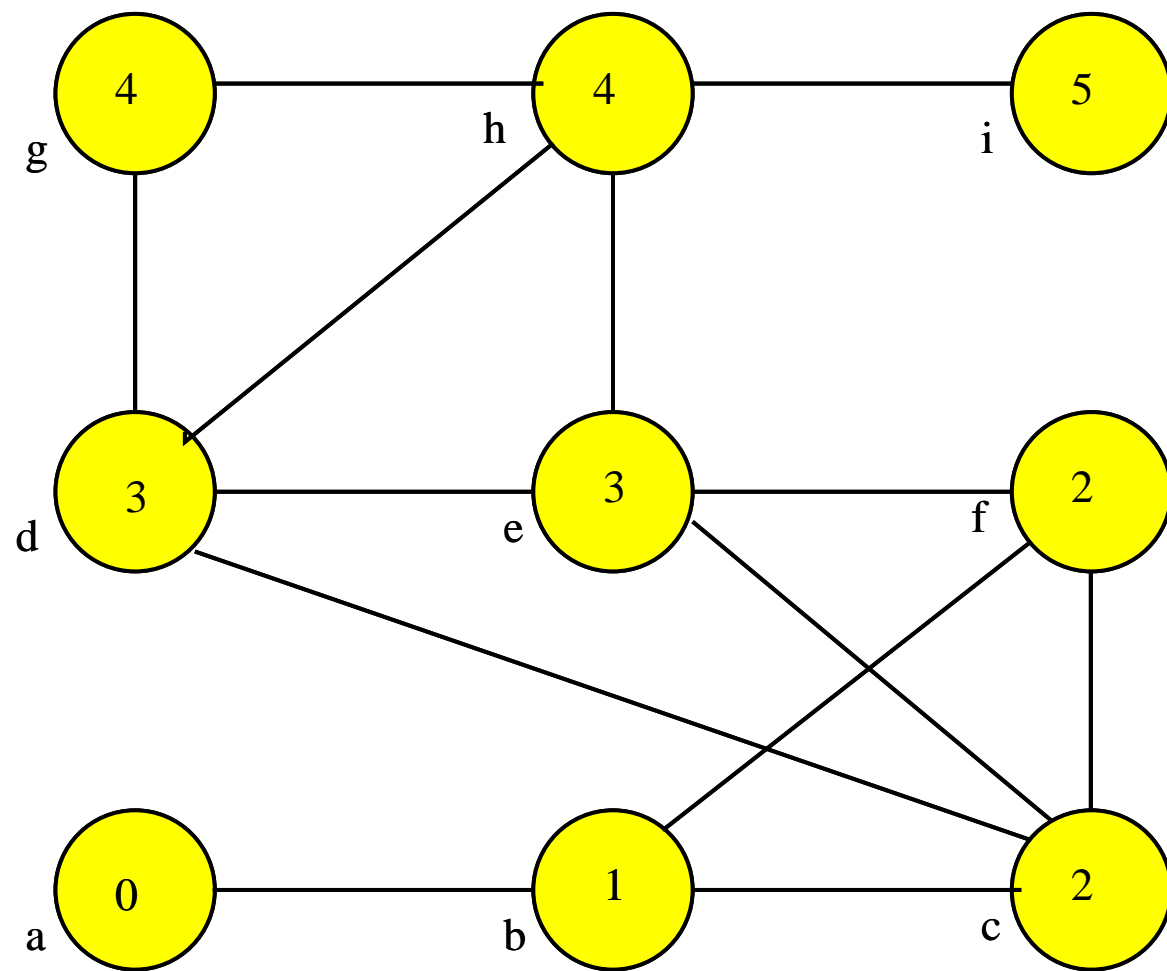












5.2 Najkrótsze ścieżki: wagi nieujemne

- przypomnijmy, że w grafie z wagami na krawędziach, najkrótsza ścieżka między dwoma wierzchołkami to ścieżka o najmniejszej wadze – nie koniecznie mająca najmniej krawędzi
- w przypadku, gdy wszystkie wagi są nieujemne do wyznaczenia najkrótszej ścieżki między dwoma ustalonymi wierzchołkami s i t stosujemy najczęściej *algorytm Dijkstry*

- podstawową ideą algorytmu jest przemieszczanie się po krawędziach grafu z wierzchołka s w kierunku wierzchołka t i cechowanie wierzchołków ich bieżącymi odległościami od wierzchołka s
- cecha wierzchołka v staje się **stała**, gdy jest równa długości najkrótszej ścieżki z s do v
- wierzchołki, które nie zostały ocechowane stałymi cechami mają cechy tymczasowe
- cechę tymczasową możemy interpretować jako długość najkrótszej z dotychczas znalezionych ścieżek z s do t .

W algorytmie posługujemy się następującymi oznaczeniami:

1. *Cecha*[v] – długość aktualnie najkrótszej ścieżki między s i v ;
2. *Poprzednik*[v] – bezpośredni poprzednik wierzchołka v na aktualnie najkrótszej ścieżce z s do v ;
3. *Tymczasowe* – zbiór wierzchołków mających aktualnie cechy tymczasowe.

Uwaga: opis algorytmu zakłada, że mamy do czynienia z digrafem; dla grafów nieskierowanych “poprzednik” oznacza po prostu “sąsiad”

Algorytm Dijkstry

1. Nadaj wierzchołkowi s cechę równą 0 ($Cecha[s] \leftarrow 0$). Pozostałym wierzchołkom v , $v \neq s$, nadaj cechę równą ∞ ($Cecha[v] := \infty$) oraz *Poprzednik* niezdefiniowany ($Poprzednik[v] \leftarrow 0$). Zdefiniuj wartości pozostałych zmiennych $Tymczasowe \leftarrow V \setminus \{s\}$, $z \leftarrow s$.
2. Wszystkim następnikom u wierzchołka z , które nie mają cechy stałej ($u \in Tymczasowe$), dla których $Cecha[u] > Cecha[z] + w_{zu}$, nadaj nowe cechy tymczasowe $Cecha[u] \leftarrow Cecha[z] + w_{zu}$. Zmień dla nich również drugą etykietę $Poprzednik[u] \leftarrow z$.

3. Spośród wierzchołków z $Tymczasowe$ wybierz jeden o najmniejszej cenie i zapisz go jako x . Nadaj x cechę stałą $Tymczasowe \leftarrow Tymczasowe - \{x\}$; $z := x$.
4. Jeżeli $z \neq t$ to wróć do kroku 2.
5. STOP — długość najkrótszej ścieżki z wierzchołka s do t wynosi $Cecha[t]$, natomiast sama ścieżka ma następującą postać :

$$(s, \dots, Poprzednik[Poprzednik[t]], Poprzednik[t], t).$$

Algorithm 0.0.1: DIJKSTRA($G = (V, W), s, t$)

for each $v \in V$

do $Cecha[v] \leftarrow \infty$; $Poprzednik[v] \leftarrow 0$

$Cecha[s] \leftarrow 0$; $Tymczasowe \leftarrow V \setminus \{s\}$; $z \leftarrow s$;

repeat

$M \leftarrow \infty$

for each $u \in \Gamma(z) \cap Tymczasowe$

do $\left\{ \begin{array}{l} \text{if } Cecha[u] > Cecha[z] + W[z, u] \\ \quad \text{then } \left\{ \begin{array}{l} Cecha[u] \leftarrow Cecha[z] + W[z, u] \\ Poprzednik[u] \leftarrow z \end{array} \right. \\ \text{if } Cecha[u] \leq M \\ \quad \text{then } x \leftarrow u; M \leftarrow Cecha[u] \end{array} \right.$

$Tymczasowe \leftarrow Tymczasowe \setminus \{x\}$

$z \leftarrow x$

until $x = t$

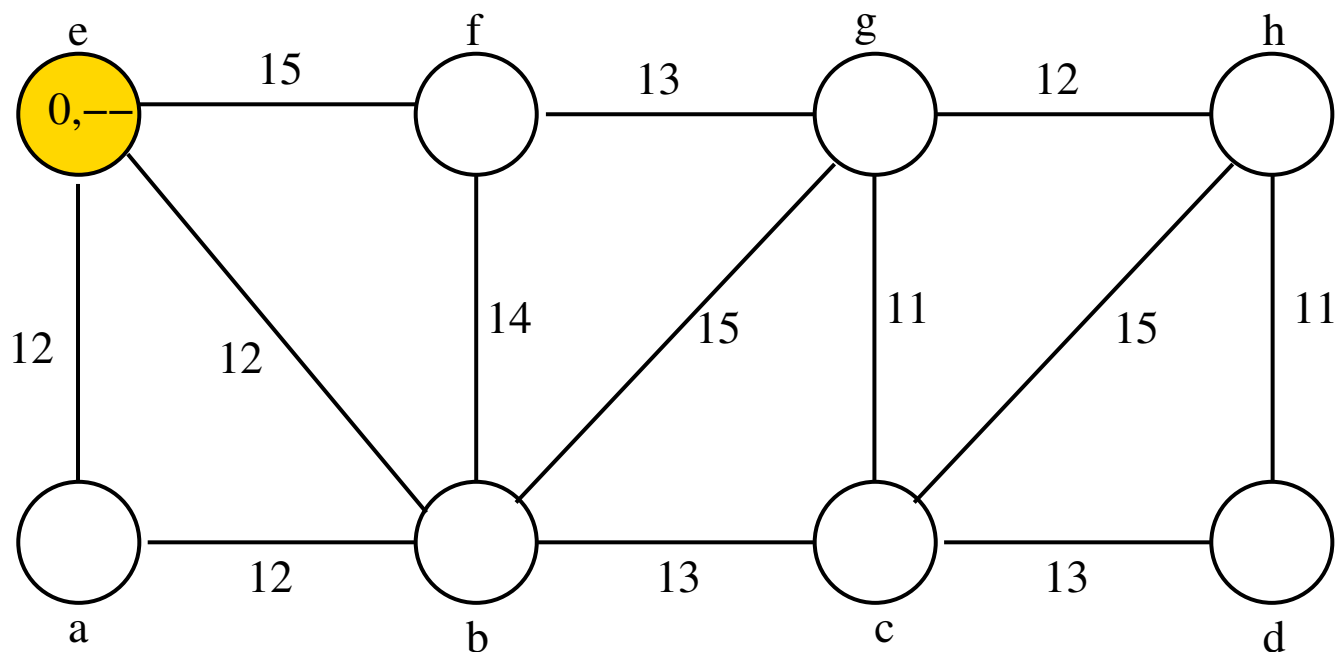
return ($Cecha[t]$)

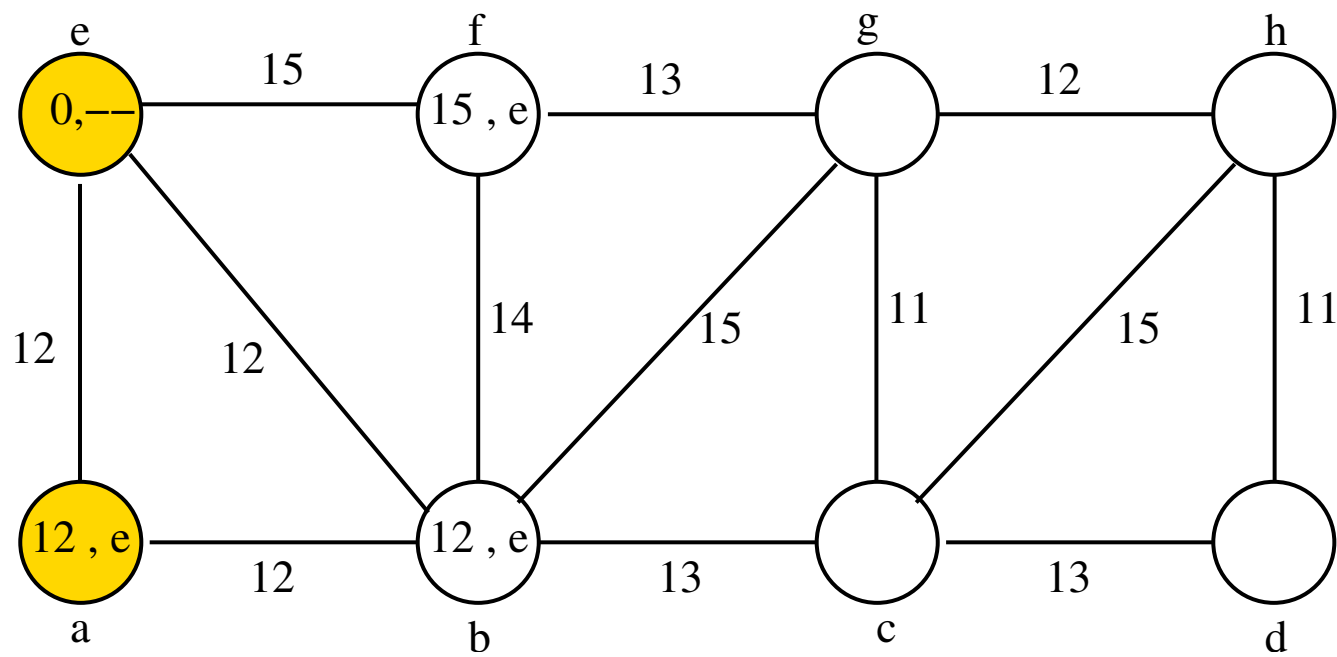
Uwaga! Jeżeli w opisie algorytmu zamienimy krok czwarty na:

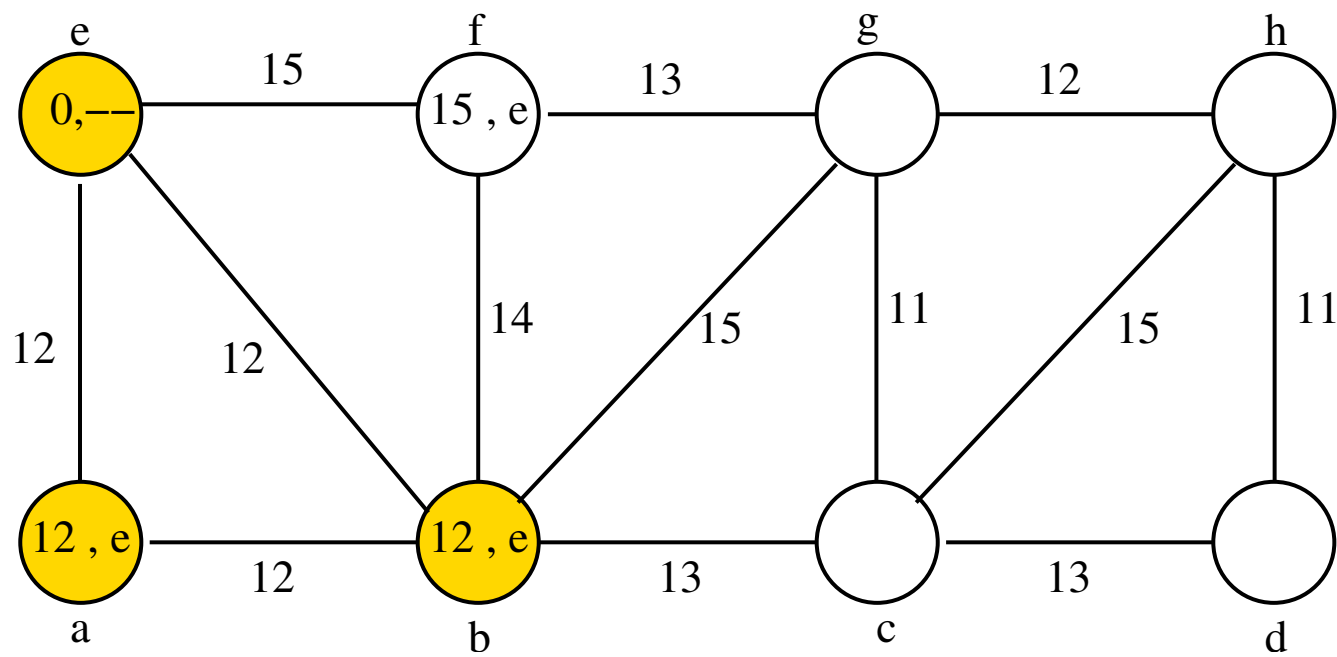
4. jeżeli $T_{\text{ymczasowe}} \neq \emptyset$ to wróć do kroku 2.

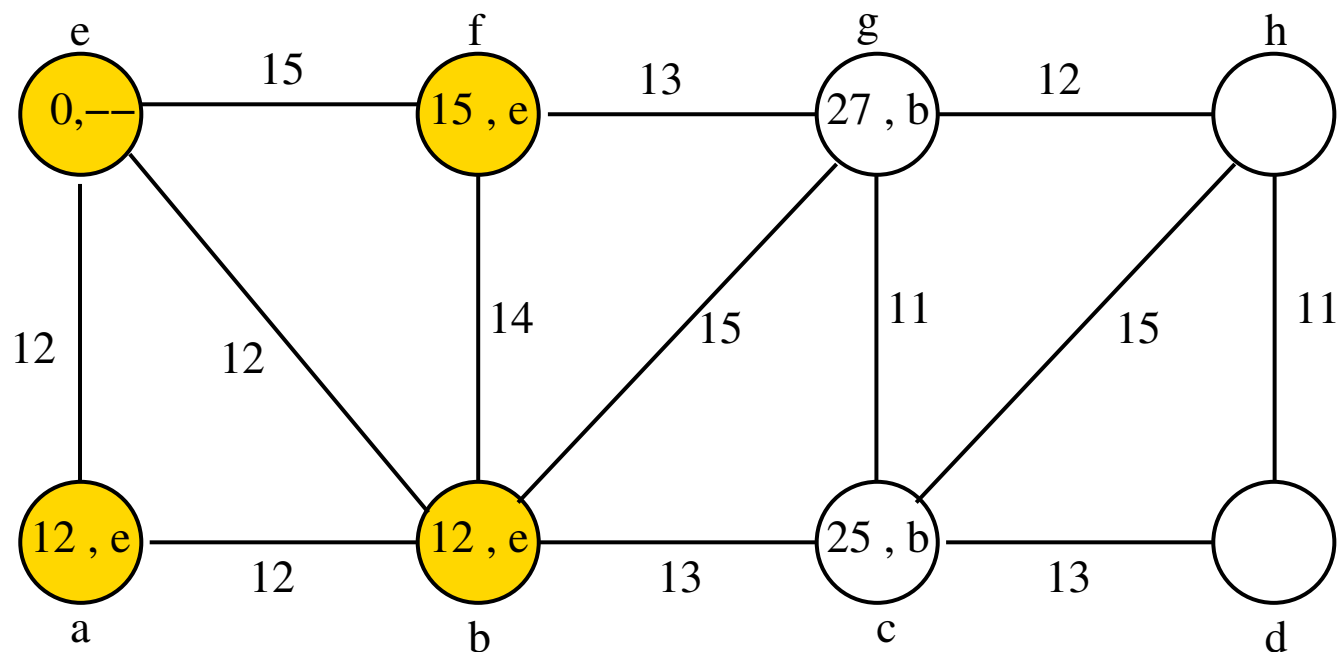
lub, równoważnie, jeżeli w pseudokodzie zamienimy linię **until** $x = t$ na **until** $T_{\text{ymczasowe}} \neq \emptyset$, to algorytm Dijkstry wyznaczy odległość s od wszystkich pozostałych wierzchołków grafu.

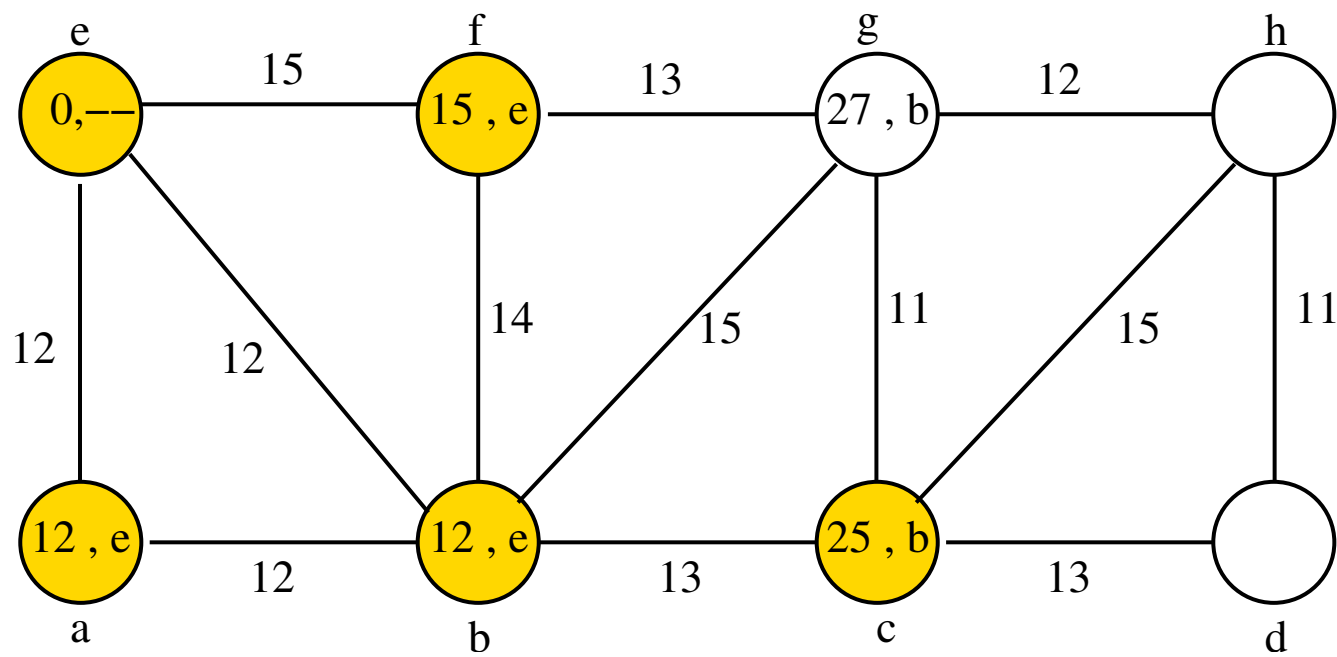
Przykład . Wyznaczyć za pomocą algorytmu Dijkstry długość najkrótszej ścieżki między wierzchołkami e i h w poniższym grafie:

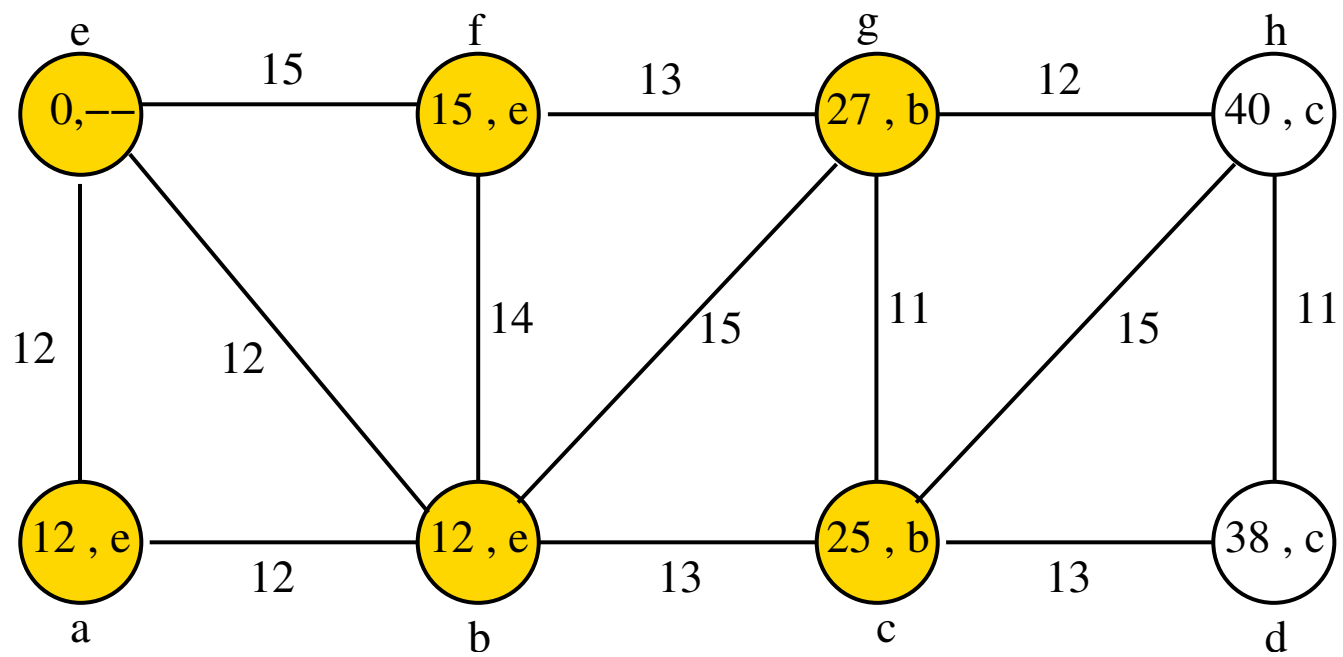


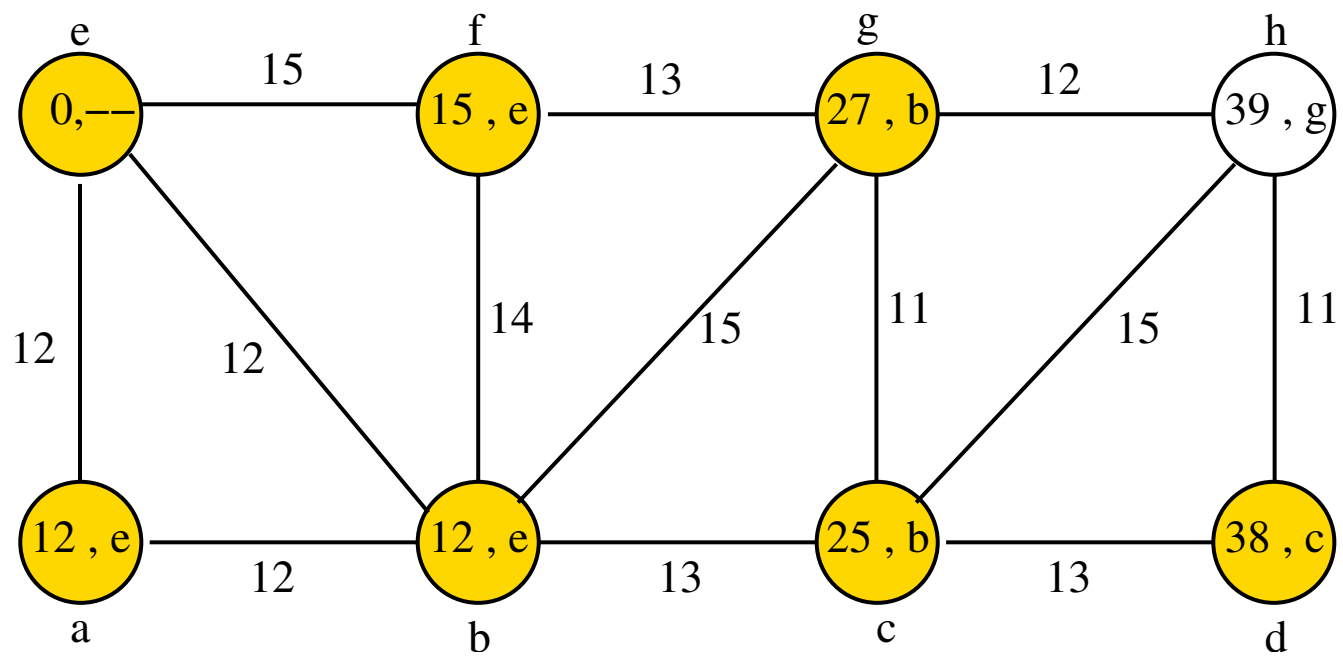


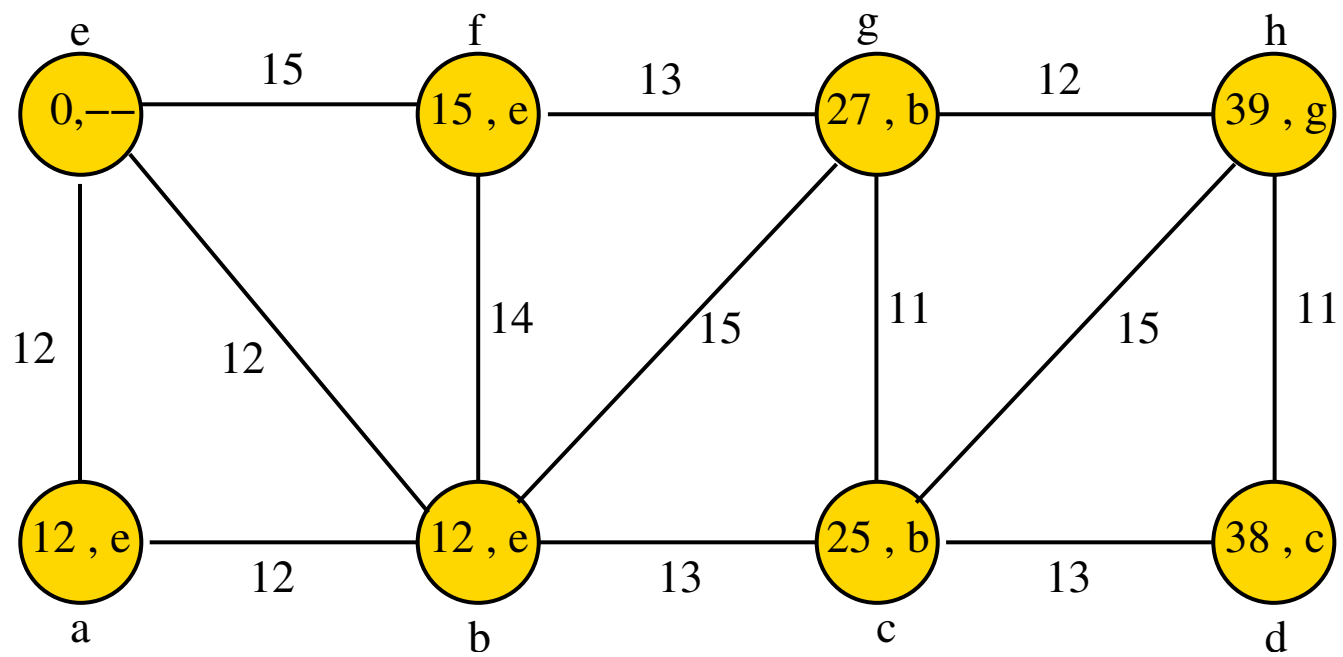


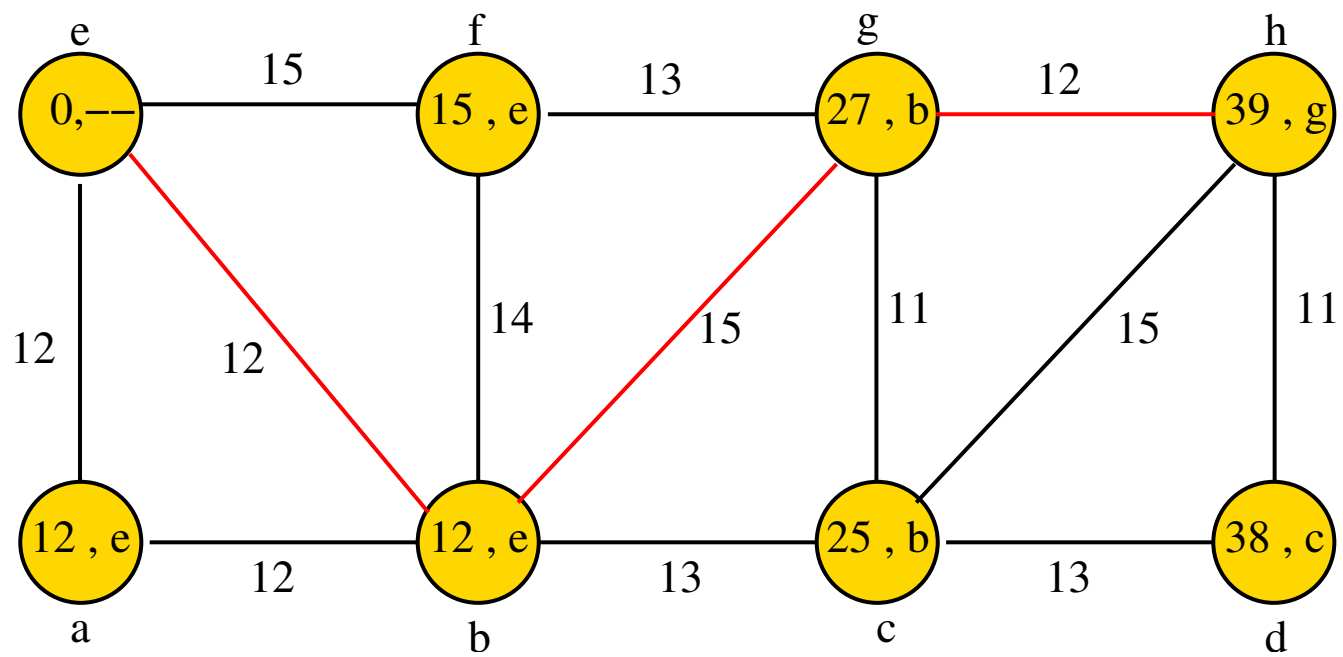












- zauważmy, że zewnętrzna pętla **repeat** w algorytmie Dijkstry może być wykonana co najwyżej $|V| - 1$ razy (z może być, co najwyżej jeden raz, pewnym wierzchołkiem z $V \setminus \{s\}$)
- maksymalny przebieg tej pętli ma miejsce wtedy, gdy wierzchołek końcowy t otrzymuje cechę stałą jako ostatni
- podczas każdego wykonania tej pętli wykonujemy $O(|V|)$ operacji
- złożoność obliczeniowa algorytmu, w obu wersjach – wyznaczenie $\rho(s, t)$ i wyznaczenie odległości s od wszystkich wierzchołków, jest rzędu $O(n^2)$, gdzie $n = |V|$

5.3 Najkrótsze ścieżki: wagi dowolne

- jeżeli wagi pewnych krawędzi grafu są ujemne to algorytm Dijkstry nie będzie działał poprawnie, ponieważ mechanizm nadawania cech stałych zakłada, że włączenie do ścieżki dodatkowych krawędzi może ją tylko wydłużyć
- w przypadku ujemnych wag, dodanie krawędzi może długość ścieżki **zmniejszyć**
- Bellman i Ford zaproponowali modyfikację algorytmu Dijkstry, polegającą na tym, że cechy tymczasowe otrzymują te wierzchołki, których cechy w ostatnim kroku zmieniły się; algorytm kończy się, gdy w którymś kroku, żadne cechy wierzchołków nie zmieniły się

- w przypadku wag ujemnych potrzebujemy dodatkowego zabezpieczenia algorytmu przed zapętleniem się
- jeżeli w grafie istnieje cykl, którego suma wag jest ujemna (“**ujemny**” cykl), to przechodząc ten cykl “w kółko” będziemy stale zmniejszać wagę ścieżki
- dla grafu z “ujemnym” cyklem problem znajdowania długości najkrótszych ścieżek jest źle postawiony!
- zauważmy, że zbadanie czy dla danego grafu problem jest dobrze czy źle postawiony, czyli sprawdzanie długości wszystkich cykli w grafie jest bardziej skomplikowane niż szukanie najkrótszych ścieżek!

- powyższy problem daje się jednak rozwiązać w bardzo prosty sposób!
- zauważmy, że jeżeli graf o n wierzchołkach nie zawiera “ujemnych” cykli, to najkrótsza ścieżka przechodzi przez każdy wierzchołek co najwyżej raz, czyli zawiera co najwyżej $n - 1$ krawędzi i związku z tym w algorytmie wystarczy wykonać co najwyżej $n - 1$ kroków
- jeżeli po wykonaniu $n - 1$ kroków mamy w grafie wierzchołki ze zmieniającymi się cechami to oznacza, że w grafie są “ujemne” cykle!

Opis algorytmu Bellmana-Forda znajdującego najkrótsze ścieżki z ustalonego wierzchołka s do wszystkich pozostałych wierzchołków (dowolne wagi krawędzi).

Oznaczenia:

$l_k(v)$ – etykieta wierzchołka v w k -tej iteracji,

$p_k(v)$ – poprzednik wierzchołka v w k -tej iteracji,

$\Gamma(v)$ – zbiór następników wierzchołka v ,

$\Gamma^{-1}(v)$ – zbiór poprzedników wierzchołka v .

Algorytm Bellmana-Forda

1. $k \leftarrow 1, S \leftarrow \Gamma(s),$

$$l_1(v) \leftarrow \begin{cases} 0, & v = s, \\ w(s, v), & v \in \Gamma(s), \\ \infty, & v \notin \Gamma(s) \cup \{s\}, \end{cases}$$

$$p_1(u) \leftarrow \begin{cases} 0, & v = s, \\ s, & v \in \Gamma(s), \\ \infty, & v \notin \Gamma(s) \cup \{s\}. \end{cases}$$

2. Dla każdego następnika v wierzchołków z S , czyli $v \in \Gamma(S)$:

$$l_{k+1}(v) \leftarrow \min\{l_k(v), \min_{u \in T_v} \{l_k(u) + w(u, v)\},$$

gdzie $T_v = \Gamma^{-1}(v) \cap S$. Dla każdego wierzchołka $v \in \Gamma(S)$, dla którego $l_{k+1}(v) = l_k(u) + w(u, v)$:

$p_{k+1}(v) \leftarrow u$, a dla pozostałych v $p_{k+1}(v) \leftarrow p_k(v)$.

Dla $v \notin \Gamma(S)$: $l_{k+1}(v) := l_k(v)$ oraz $p_{k+1}(v) := p_k(v)$.

Zauważmy, że zbiór S zawiera wszystkie wierzchołki, do których aktualnie najkrótsze ścieżki z s składają się z k łuków. Zbiór T_v składa się z wierzchołków, do których najkrótsze ścieżki z s składają się z k łuków i których następnikiem jest v .

3. (a) Jeżeli $k \leq n - 1$ oraz $l_{k+1}(v) = l_k(v)$ dla każdego v , to STOP – znaleźliśmy wszystkie szukane odległości.
- (b) Jeżeli $k < n - 1$ oraz $l^{k+1}(v) \neq l^k(v)$ dla pewnego wierzchołka v , to przejdź do kroku 4.
- (c) Jeżeli $k = n - 1$ oraz $l^{k+1}(v) \neq l^k(v)$ dla pewnego wierzchołka v , to STOP – graf ma cykle o ujemnych sumach wag.
4. $S \leftarrow \{v; l^{k+1}(v) \neq l^k(v)\}$, $k \leftarrow k + 1$ i przejdź do kroku 2. Zbiór S zawiera wierzchołki, do których najkrótsze ścieżki z s mają $k + 1$ łuków.

- zauważmy, że nie trzeba pamiętać wszystkich wektorów l_k
- w każdej iteracji potrzebne są tylko aktualnie tworzone (z indeksem $k + 1$) oraz te z poprzedniej iteracji (z indeksem k)
- dlatego w zamieszczonym poniżej pseudokodzie algorytmu BELLMAN-FORD używamy wektora *CechaStara*, dla oznaczenia wartości z poprzedniej iteracji i *Cecha* dla wartości z aktualnej iteracji (dla wektora p wystarczy tylko jeden wektor - *Poprzednik*)

Algorithm 0.0.1: BELLMAN-FORD($G = (V, E, W), s$)

for each $v \in V$

do $CechaStara[v] \leftarrow \infty$; $Poprzednik[v] \leftarrow 0$

$CechaStara[s] \leftarrow 0$; $S \leftarrow \{s\}$; $k \leftarrow 0$;

repeat

$k \leftarrow k + 1$; $Cecha \leftarrow CechaStara$

for each $u \in S$

do for each $v \in \Gamma(u)$

do if $Cecha[v] > CechaStara[u] + W[v, u]$

then $\begin{cases} Cecha[v] \leftarrow CechaStara[u] + W[v, u] \\ Poprzednik[v] \leftarrow u \end{cases}$

$S \leftarrow \{v \in V : Cecha[v] \neq CechaStara[v]\}$

$CechaStara \leftarrow Cecha$

until $S = \emptyset \vee k = \nu - 1$

if $S \neq \emptyset$

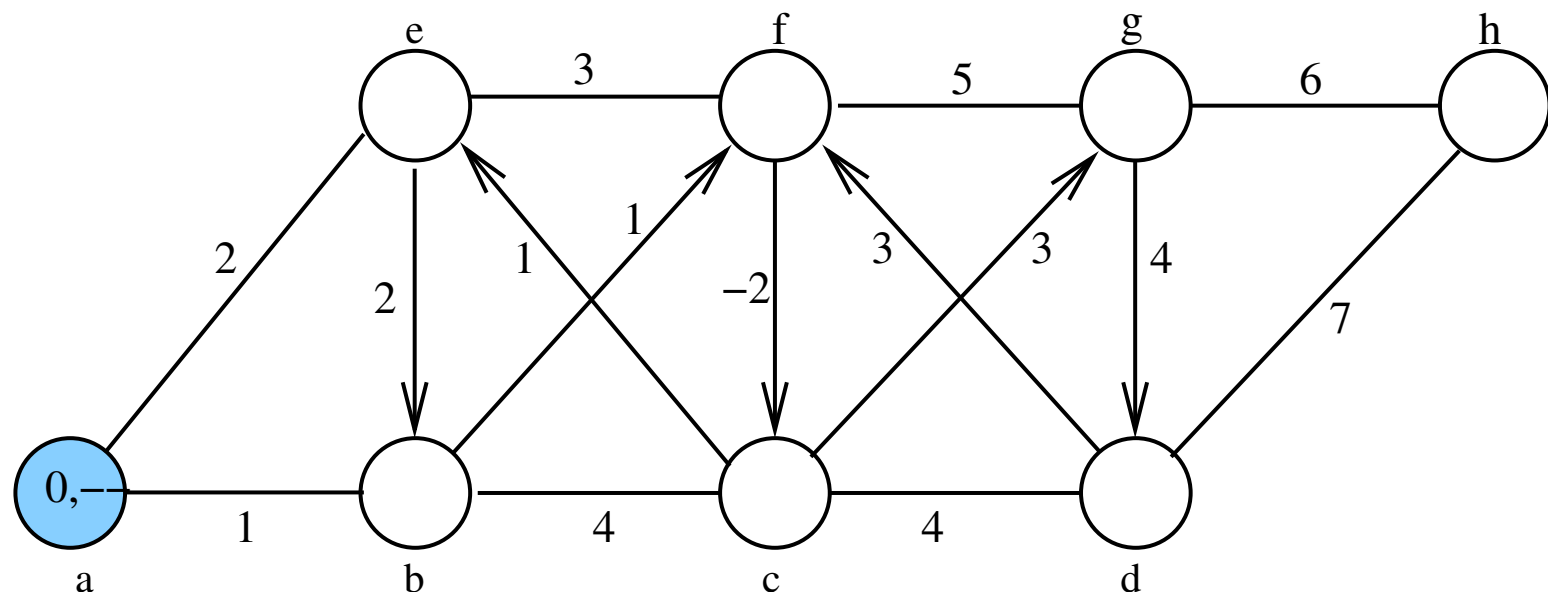
then output („Graf zawiera cykl o ujemnej sumie wag”)

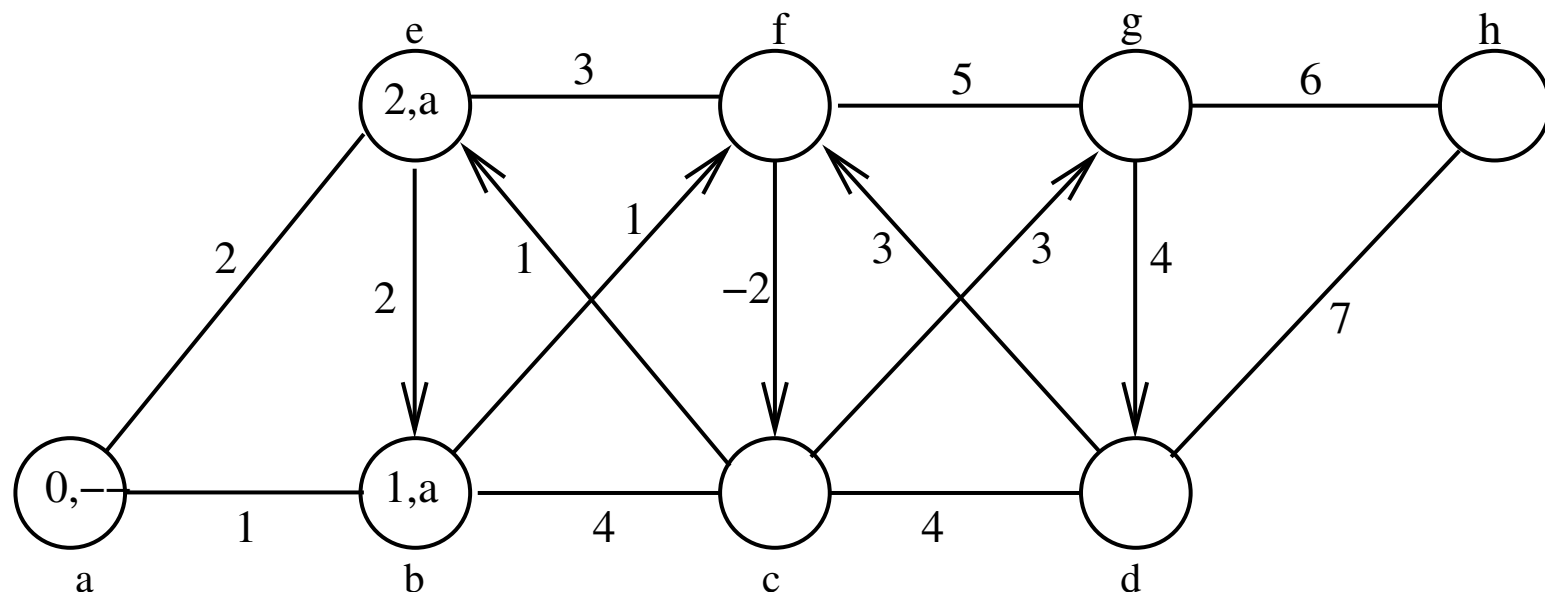
return ($Cecha$)

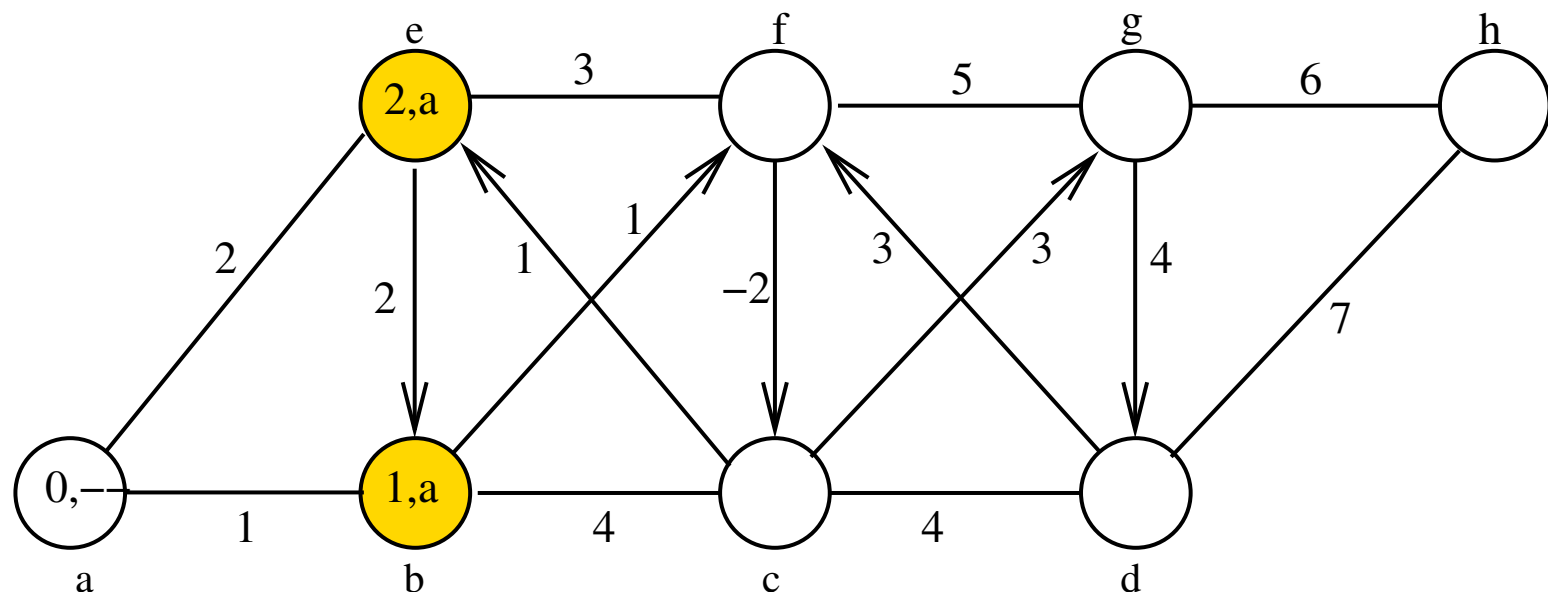
- ponieważ pętla **repeat** wykonywana jest co najwyżej $n - 1$ razy, a zakres pętli ma złożoność $O(n^2)$, więc algorytm Bellmana-Forda ma złożoność obliczeniową $O(n^3)$

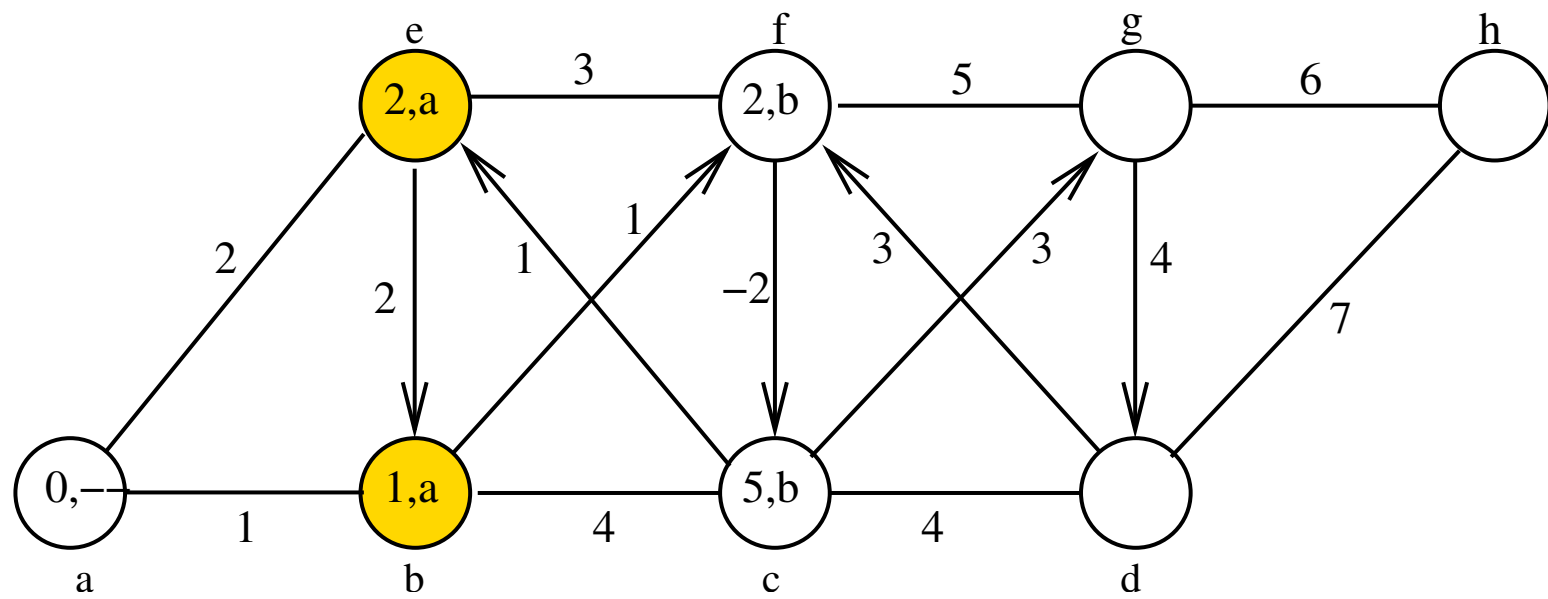
Przykład . Wyznaczyć długości najkrótszych ścieżek między wierzchołkiem a a pozostałymi wierzchołkami grafu o następującej macierzy wag:

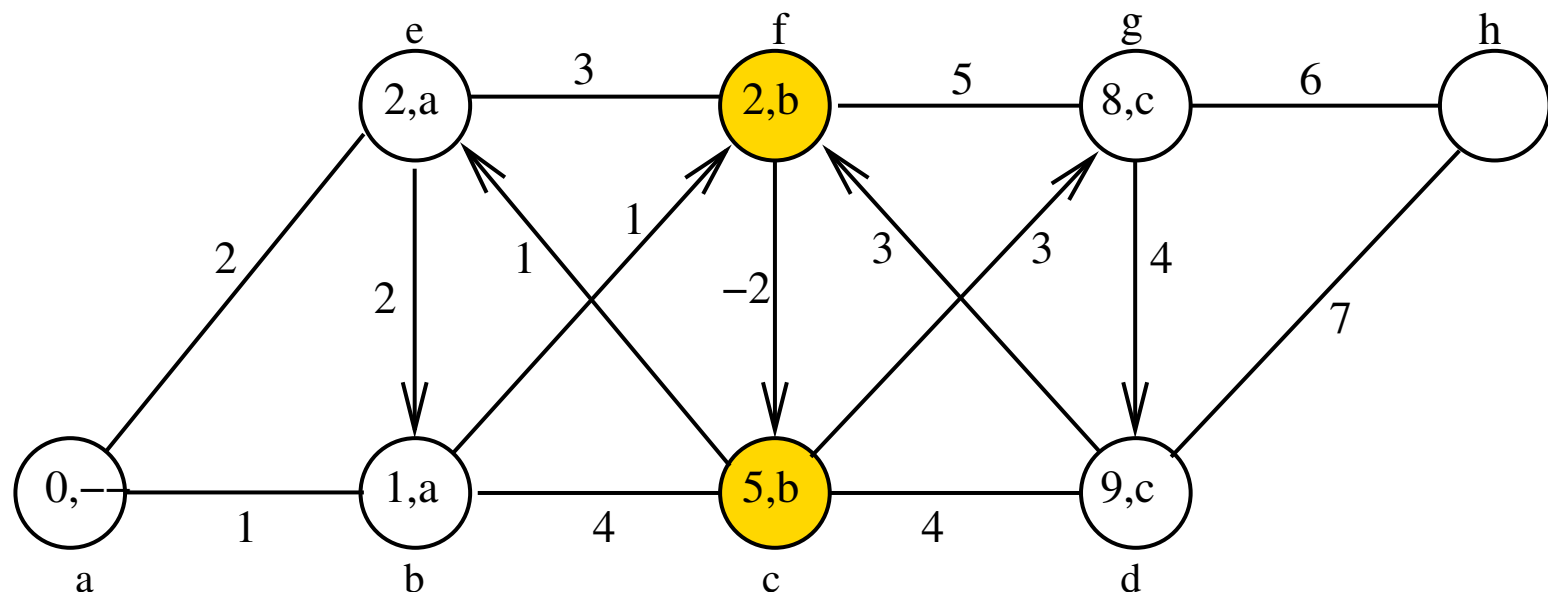
$$W = \begin{matrix} & \begin{matrix} a & b & c & d & e & f & g & h \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{matrix} & \begin{pmatrix} 0 & 1 & \infty & \infty & 2 & \infty & \infty & \infty \\ 1 & 0 & 4 & \infty & \infty & 1 & \infty & \infty \\ \infty & 4 & 0 & 4 & 3 & \infty & 3 & \infty \\ \infty & \infty & 4 & 0 & \infty & 3 & 4 & 7 \\ 2 & 2 & \infty & \infty & 0 & 3 & \infty & \infty \\ \infty & \infty & -2 & 3 & \infty & 0 & 5 & \infty \\ \infty & \infty & \infty & 4 & \infty & 5 & 0 & 6 \\ \infty & \infty & \infty & 7 & \infty & \infty & 6 & 0 \end{pmatrix} \end{pmatrix} .$$

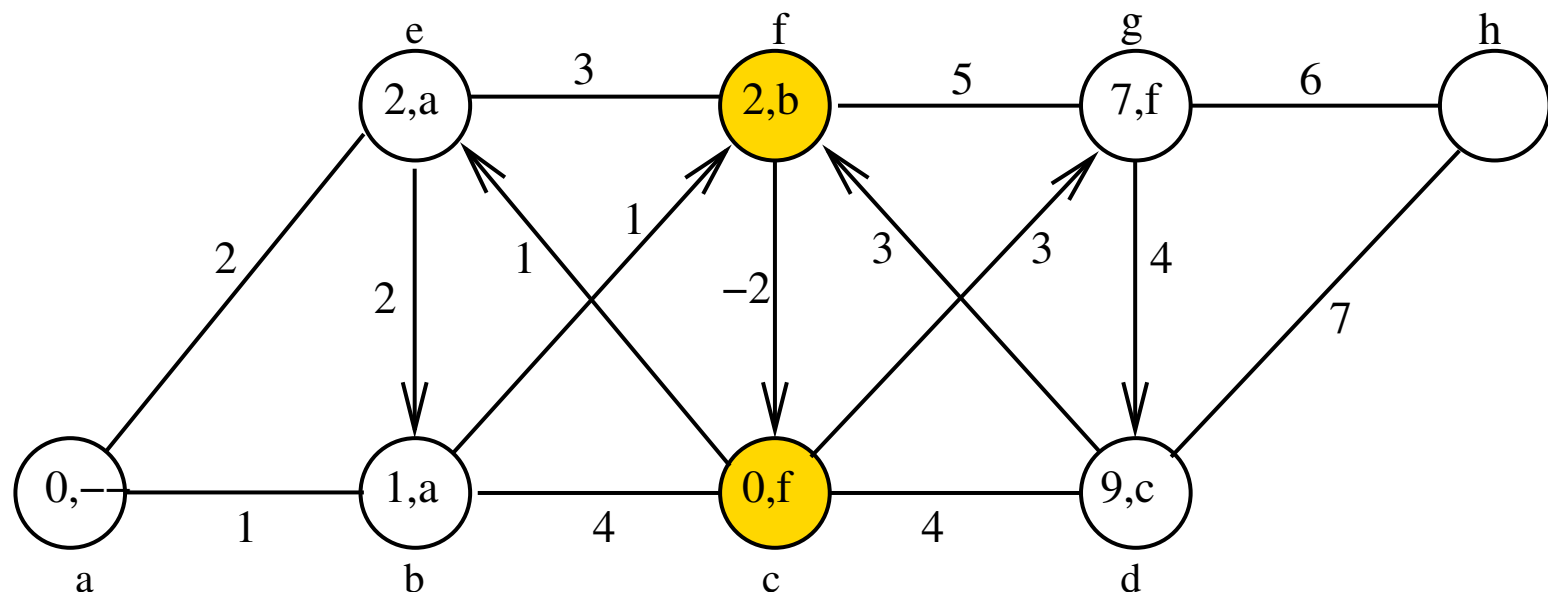


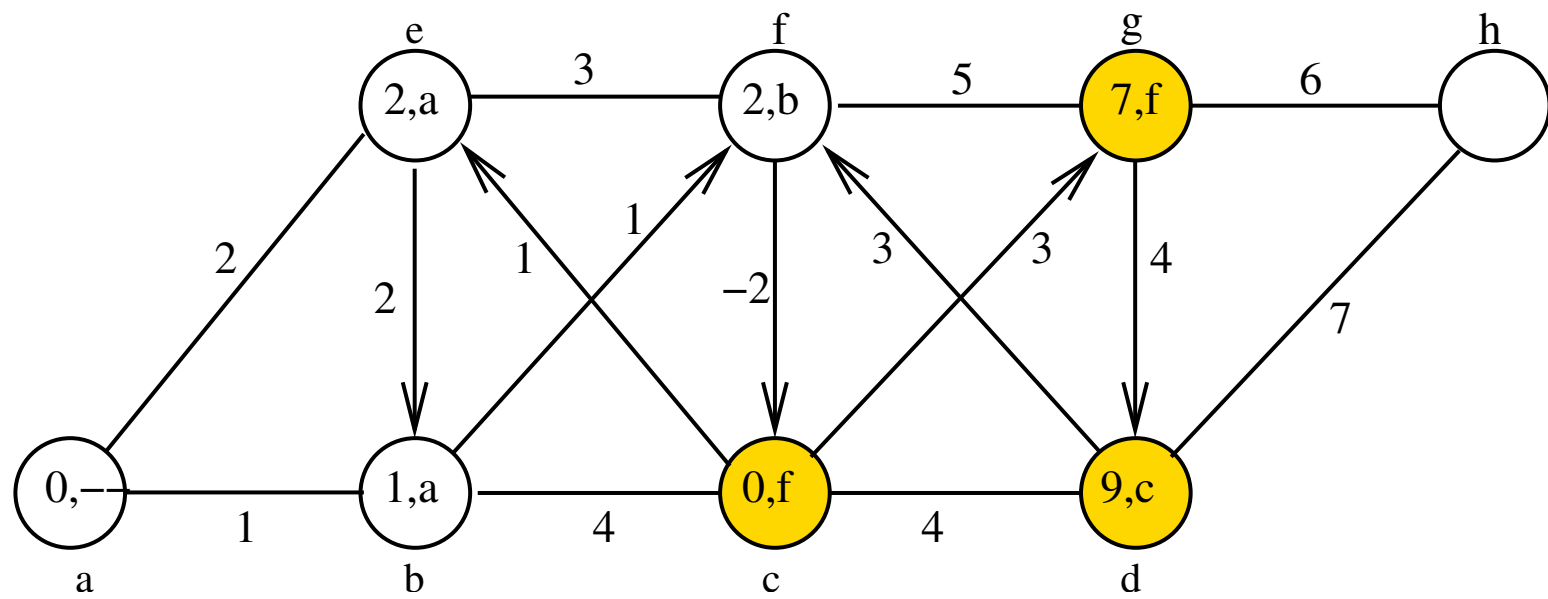


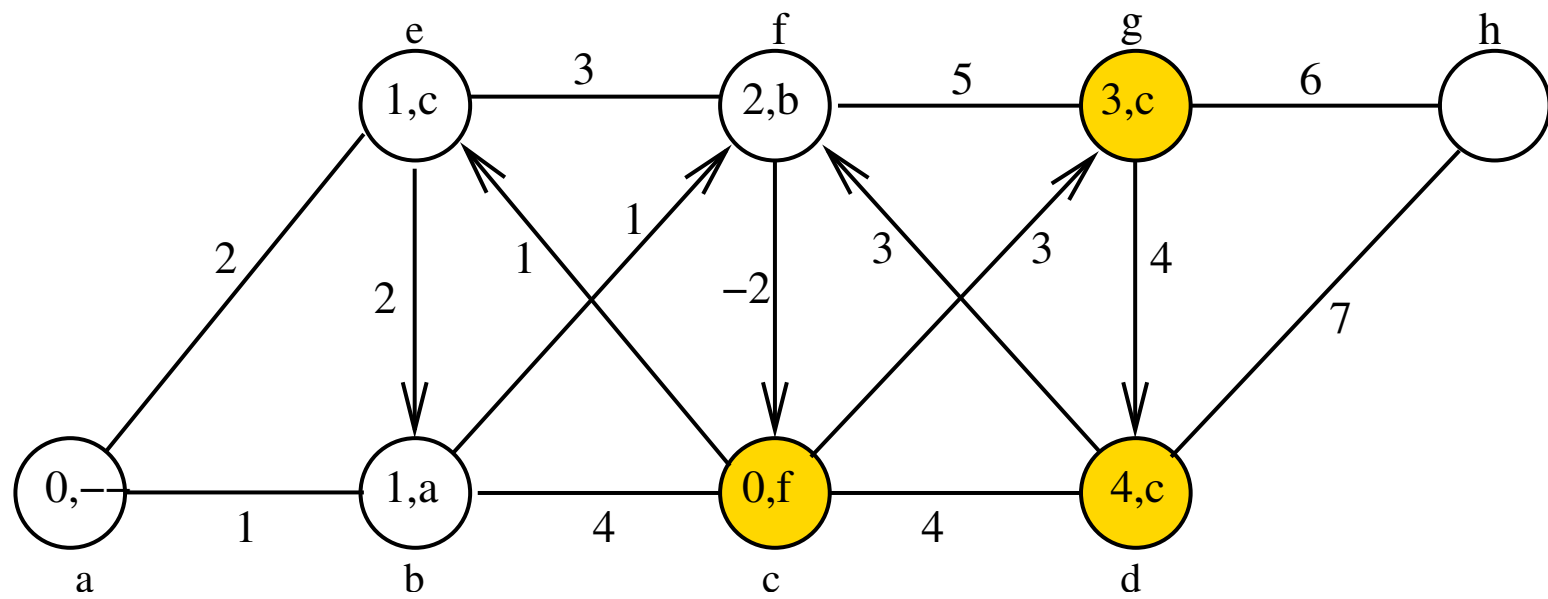


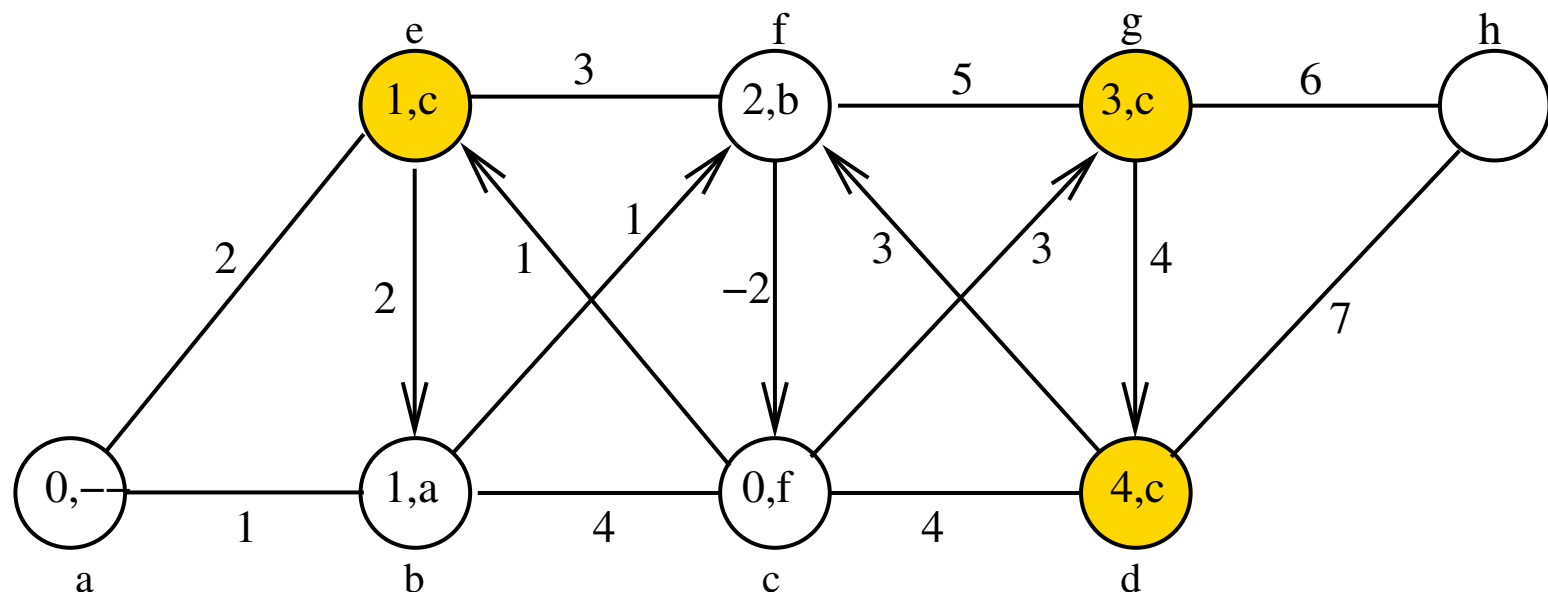


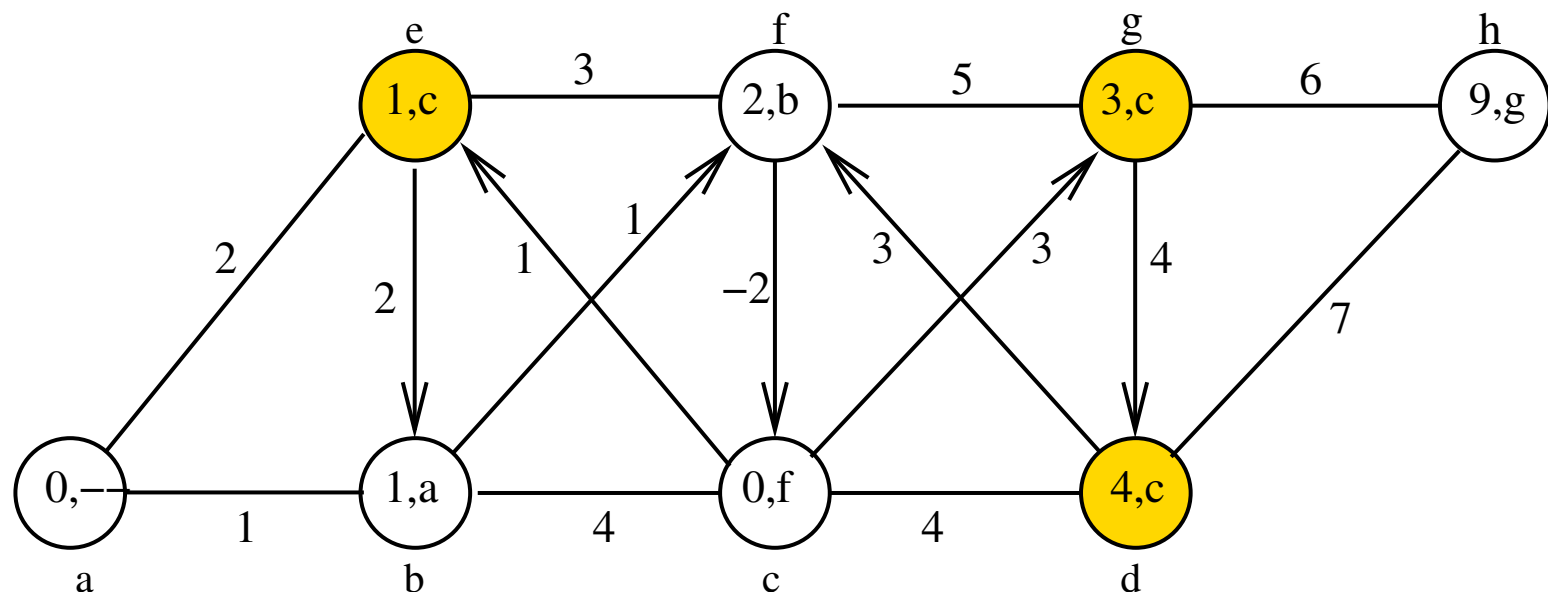


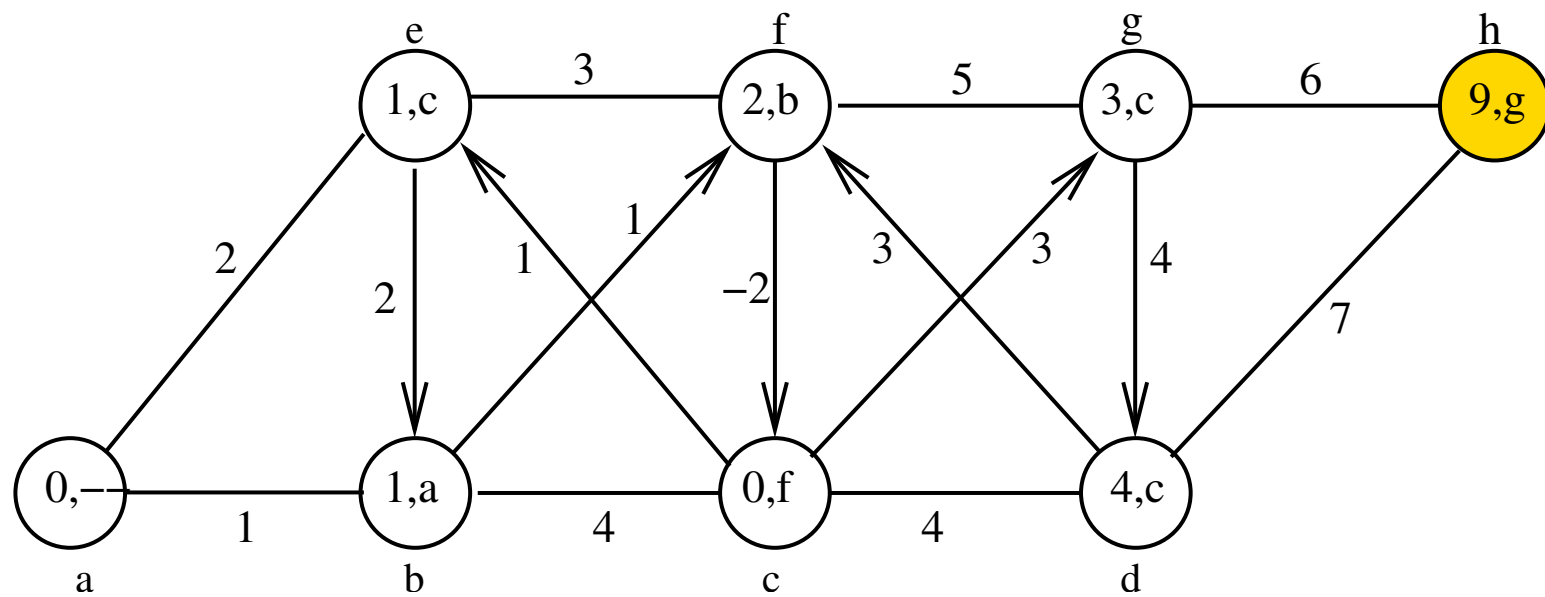


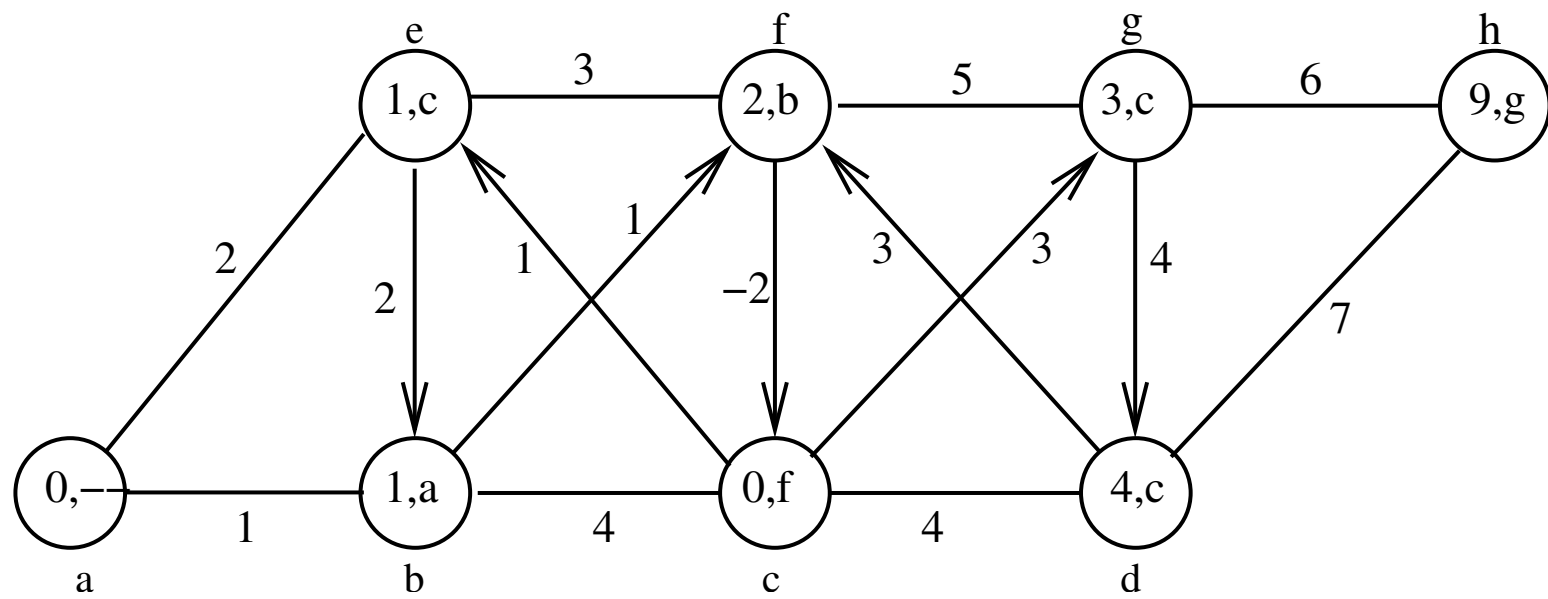












- znalezienie najkrótszych ścieżek między **wszystkimi** $\binom{n}{2}$ **parami wierzchołków** w grafie $G = (V, E)$ z **nieujemnymi wagami**, sprowadza się do n -krotnego zastosowania zmodyfikowanego algorytmu Dijkstry, znajdującego dla każdego wierzchołka grafu osobno najkrótsze ścieżki do wszystkich pozostałych wierzchołków
- ponieważ złożoność algorytmu Dijkstry wynosi $O(n^2)$, zatem dla **grafu z nieujemnymi wagami** złożoność takiej procedury jest rzędu $O(n^3)$
- w przypadku gdy w grafie występują **ujemne wagi** n -krotne zastosowanie algorytmu Bellmana-Forda daje złożoność rzędu $O(n^4)$
- **algorytm Floyd** o złożoności obliczeniowej $O(n^3)$

Algorytm Floyda

- Rozpoczynając z macierzą wag $W = (w_{ij})$ wymiaru $n \times n$, która reprezentuje długości bezpośrednich połączeń między wierzchołkami $V = \{1, 2, \dots, n\}$ w grafie, konstruowany jest ciąg macierzy

$$W^{(1)}, W^{(2)}, \dots, W^{(n)}.$$

Element $w_{ij}^{(k)}$ macierzy $W^{(k)}$ jest długością najkrótszej ścieżki spośród wszystkich ścieżek z wierzchołka i do wierzchołka j , których wierzchołki pośrednie należą do zbioru $\{1, 2, \dots, k\}$

- macierz $W^{(k)}$ tworzymy z macierzy $W^{(k-1)}$ przy czym:

$$w_{ij}^{(0)} = w_{ij},$$

$$w_{ij}^{(k)} = \min\{w_{ij}^{(k-1)}, w_{ik}^{(k-1)} + w_{kj}^{(k-1)}\} \quad \text{dla } k = 1, 2, \dots, n.$$

Zauważmy, że powyższy algorytm wyznacza długości najkrótszych ścieżek między każdą parą wierzchołków i, j a nie same drogi. Aby je wyznaczyć posłużymy się dodatkowo budowaną macierzą $P = (p_{ij})$ wymiaru $n \times n$.

Element p_{ij} jest macierzy P przedostatnim wierzchołkiem na najkrótszej drodze z i do j . Jeśli ta droga ma, na przykład, postać $(i, v_1, v_2, \dots, v_q, j)$, to kolejne wierzchołki możemy otrzymać z macierzy P :

$$i = p_{iv_1}, \dots, v_{q-2} = p_{iv_{q-1}}, v_{q-1} = p_{iv_q}, v_q = p_{ij}, j.$$

Macierz P tworzymy w następujący sposób:

- na początku ustalamy, że jeżeli $w_{ij} = \infty$, to p_{ij} jest równe 0, w przeciwnym przypadku p_{ij} jest równe i . W k -tej iteracji, jeśli wierzchołek k został włączony do ścieżki z i do j (tzn. $w_{ij} > w_{ik} + w_{kj}$) za p_{ij} przyjmujemy p_{kj} (zmienia się poprzednik wierzchołka j).

Algorithm 0.0.1: FLOYD($G = (V, E, W)$)

$CechaStara \leftarrow W$

for each $v \in V$

do for each $w \in V$

do if $v = w \vee W[v, w] = \infty$

then $PStare[v, w] \leftarrow 0$

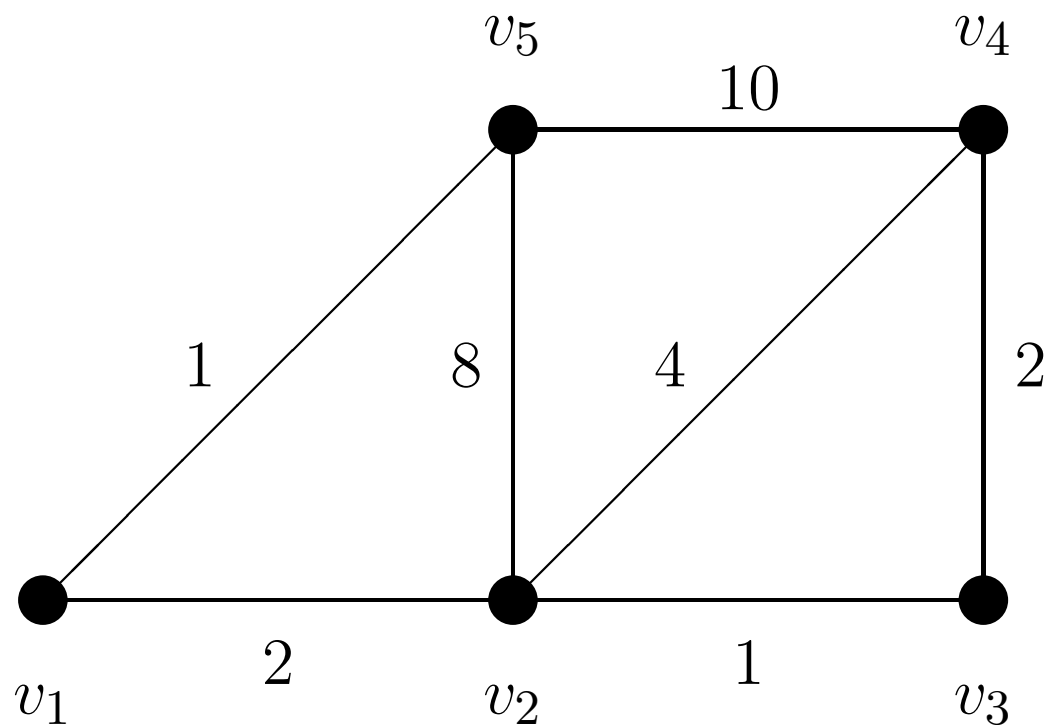
else $PStare[v, w] \leftarrow v$

for $k = 1$ **to** $|V|$

do $\left\{ \begin{array}{l} Cecha \leftarrow CechaStara; P \leftarrow PStare \\ \textbf{for each } v \in V \\ \quad \textbf{do for each } w \in V \\ \quad \textbf{do if } CechaStara[v, w] > CechStara[v, k] + CechaStara[k, w] \\ \quad \textbf{then } \left\{ \begin{array}{l} Cecha[v, w] \leftarrow CechStara[v, k] + CechaStara[k, w] \\ P[v, w] \leftarrow PStare[k, w] \end{array} \right. \\ \textbf{for each } v \in V \\ \quad \textbf{do if } Cech[v, v] < 0 \\ \quad \textbf{then } \left\{ \begin{array}{l} \textbf{output (Graf zawiera cykl o ujemnej sumie wag)} \\ \textbf{stop} \end{array} \right. \end{array} \right.$

return $(Cecha, P)$

Przykład . Znaleźć najkrótszą drogę między każdą parą wierzchołków w grafie przedstawionym na poniższym rysunku:



Macierz wag tego grafu oraz początkowa macierz ścieżek P mają postać:

$$W^{(0)} = \begin{pmatrix} 0 & 2 & \infty & \infty & 1 \\ 2 & 0 & 1 & 4 & 8 \\ \infty & 1 & 0 & 2 & \infty \\ \infty & 4 & 2 & 0 & 10 \\ 1 & 8 & \infty & 10 & 0 \end{pmatrix} \quad \text{i} \quad P^{(0)} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 2 & 0 & 2 & 2 & 2 \\ 0 & 3 & 0 & 3 & 0 \\ 0 & 4 & 4 & 0 & 4 \\ 5 & 5 & 0 & 5 & 0 \end{pmatrix} .$$

Ponieważ w macierzy $W^{(1)}$ zmienił się element w_{25} (bo $w_{25}^{(0)} > w_{21}^{(0)} + w_{15}^{(0)}$, czyli $8 > 2 + 1$) oraz element w_{52} , dlatego też w macierzy $P^{(1)}$ zamieniamy dwa elementy: $p_{25} \leftarrow p_{15}$ oraz $p_{52} \leftarrow p_{12}$:

$$W^{(1)} = \begin{pmatrix} 0 & 2 & \infty & \infty & 1 \\ 2 & 0 & 1 & 4 & 3 \\ \infty & 1 & 0 & 2 & \infty \\ \infty & 4 & 2 & 0 & 10 \\ 1 & 3 & \infty & 10 & 0 \end{pmatrix} \quad \text{i} \quad P^{(1)} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 2 & 0 & 2 & 2 & 1 \\ 0 & 3 & 0 & 3 & 0 \\ 0 & 4 & 4 & 0 & 4 \\ 5 & 1 & 0 & 5 & 0 \end{pmatrix}.$$

Tworzymy macierz $W^{(2)}$. Zmieniły się elementy w_{13} , w_{14} , w_{31} , w_{35} , w_{41} , w_{53} , w_{54} , zatem macierze $W^{(2)}$ i $P^{(2)}$ wyglądają następująco:

$$W^{(2)} = \begin{pmatrix} 0 & 2 & 3 & 6 & 1 \\ 2 & 0 & 1 & 4 & 3 \\ 3 & 1 & 0 & 2 & 4 \\ 6 & 4 & 2 & 0 & 7 \\ 1 & 3 & 4 & 7 & 0 \end{pmatrix} \quad \text{i} \quad P^{(2)} = \begin{pmatrix} 0 & 1 & 2 & 2 & 1 \\ 2 & 0 & 2 & 2 & 1 \\ 2 & 3 & 0 & 3 & 1 \\ 2 & 4 & 4 & 0 & 1 \\ 5 & 1 & 2 & 2 & 0 \end{pmatrix}.$$

Budujemy macierz $W^{(3)}$ i $P^{(3)}$:

$$W^{(3)} = \begin{pmatrix} 0 & 2 & 3 & 5 & 1 \\ 2 & 0 & 1 & 3 & 3 \\ 3 & 1 & 0 & 2 & 4 \\ 5 & 3 & 2 & 0 & 6 \\ 1 & 3 & 4 & 6 & 0 \end{pmatrix} \quad \textbf{i} \quad P^{(3)} = \begin{pmatrix} 0 & 1 & 2 & 3 & 1 \\ 2 & 0 & 2 & 3 & 1 \\ 2 & 3 & 0 & 3 & 1 \\ 2 & 3 & 4 & 0 & 1 \\ 5 & 1 & 2 & 3 & 0 \end{pmatrix} .$$

- Zauważmy, że $W^{(3)} = W^{(4)} = W^{(5)}$, zatem $P^{(3)} = P^{(4)} = P^{(5)}$. Utworzyliśmy więc potrzebny ciąg macierzy W oraz P co kończy algorytm.
- Z macierzy tych możemy dla każdej pary wierzchołków “odczytać” najkrótszą ścieżkę łączącą te wierzchołki (nie koniecznie jedyną). Przykładowo, najkrótsza droga z wierzchołka v_4 do v_5 ma długość 6 i jest postaci $p_{43}p_{42}p_{41}p_{45}v_5$, czyli jest to ścieżka $v_5v_3v_2v_1v_5$. Konstruujemy ją od końca, ponieważ wiemy, że jej przedostatnim wierzchołkiem jest p_{45} czyli v_1 .
- Algorytm ten jak już wyżej wspomniano potrzebuje $O(n^3)$ operacji (bez względu na gęstość rozpatrywanego grafu).

Podsumowanie

- Wybór algorytmu spośród algorytmów **Dijkstry**, **Bellmana-Forda** i **Floyda** zależy oczywiście od rodzaju problemu, który mamy rozwiązać.
- Algorytm **Dijkstry** jest bardzo efektywny, może być stosowany do znajdowania najkrótszych dróg między każdą parą wierzchołków, o ile graf nie zawiera krawędzi z ujemnymi wagami.
- Algorytm **Floyda** stosujemy, gdy chcemy znaleźć najkrótsze drogi między każdą parą wierzchołków, a w grafie znajdują się krawędzie o ujemnych wagach.
- Algorytm **Bellmana-Forda** stosujemy gdy w grafie są ujemne wagi a jeden z wierzchołków jest zawsze początkiem ścieżki (źródłem, bazą).

6. Rozpięte drzewa

6.1 Definicje i twierdzenia

Definicja (Drzewo rozpięte). *Rozpiętym drzewem grafu G nazywamy spójny i acykliczny podgraf grafu G zawierający wszystkie jego wierzchołki.*

Twierdzenie . *Każdy graf spójny zawiera drzewo rozpięte.*

Twierdzenie . *W grafie spójnym $G = (V, E)$ krawędź $e \in E$ jest krawędzią cięcia wtedy i tylko wtedy, gdy e należy do każdego drzewa rozpiętego grafu G .*

Przypomnijmy, że krawędź e grafu G została **ściągnięta**, jeżeli scaliliśmy jej końce, a otrzymaną z niej pętlę usunęliśmy. Otrzymany w ten sposób graf oznaczamy będziemy $G \cdot e$.

Twierdzenie . Niech $G = (V, E)$ będzie grafem, a $\tau(G)$ oznacza liczbę rozpiętych drzew grafu G . Jeżeli $e \in E$ nie jest pętlą w grafie G , to

$$\tau(G) = \tau(G - e) + \tau(G \cdot e).$$

Twierdzenie (Twierdzenie Cayleya).

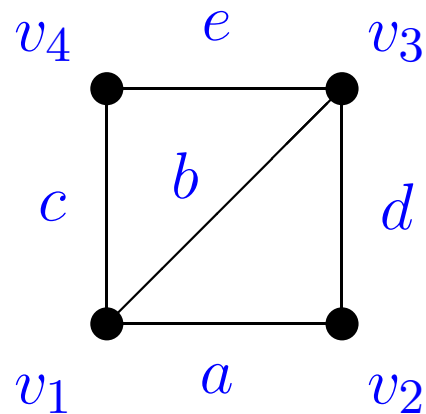
$$\tau(K_n) = n^{n-2}.$$

6.2 Generowanie drzew rozpiętych grafu

- generowanie drzewa rozpiętego danego grafu spójnego G na n wierzchołkach można realizować biorąc kolejno krawędzie z pewnej listy i akceptując je po sprawdzeniu, czy nie tworzą one cyklu z dotychczas zaakceptowanymi krawędziami
- zauważmy, że do czasu kiedy zaakceptowanych krawędzi będzie $n - 1$, czyli do czasu otrzymania drzewa, rozpięty podgraf grafu G generowany zbiorem zaakceptowanych krawędzi jest lasem
- generowanie wszystkich drzew rozpiętych danego grafu jest złożone obliczeniowo, ponieważ drzew rozpiętych może być bardzo dużo (patrz twierdzenie Cayleya)

- uporządkujemy wszystkie krawędzie grafu w dowolny sposób tworząc listę krawędzi (e_1, e_2, \dots, e_m) , gdzie $m = |E|$
- każdy podzbiór krawędzi będziemy przedstawiać w postaci listy uporządkowanej, zgodnie z porządkiem w liście krawędzi; do generowania wszystkich drzew rozpiętych wykorzystamy **algorytm z nawrotami**

Przykład . Znajdź wszystkie drzewa rozpięte grafu:



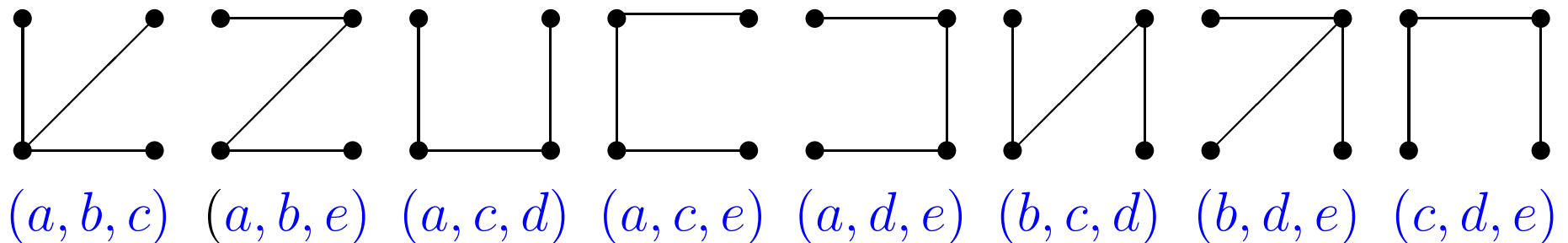
- łatwo sprawdzić, że ten graf zawiera 8 różnych drzew rozpiętych
- utwórzmy najpierw listę krawędzi: (a, b, c, d, e)
- każde drzewo rozpięte tego grafu ma trzy krawędzie
- tworzymy, zgodnie z algorytmem z nawrotami, kolejne listy; przekreślenie listy oznacza, że krawędzie te zawierają cykl i trzeba się wycofać, natomiast listy w ramkach natomiast oznaczają drzewa rozpięte, a wszystkie listy nie przekreślone oznaczają wszystkie rozpięte lasy naszego grafu
- listy (drzewa) są wygenerowane w porządku leksykograficznym (alfabetycznym)

- 1 $()$, (a) , (a, b) , (a, b, c) . Mamy pierwsze drzewo rozpięte!
- 2 Usuwamy ostatnią krawędź i kontynuujemy: ~~(a, b, d)~~ zawiera cykl, usuwamy więc d i generujemy następną listę: (a, b, e) . Otrzymaliśmy drugie drzewo rozpięte.
- 3 Usuwamy krawędź e , nie możemy jednak kontynuować bo e jest ostatnią krawędzią na liście. Usuwamy więc kolejną krawędź – b i generujemy dalej: (a, c) , (a, c, d) . Mamy kolejne drzewo.
- 4 Usuwamy ostatnią krawędź d i kontynuujemy: (a, c, e) . Znaleźliśmy kolejne drzewo.

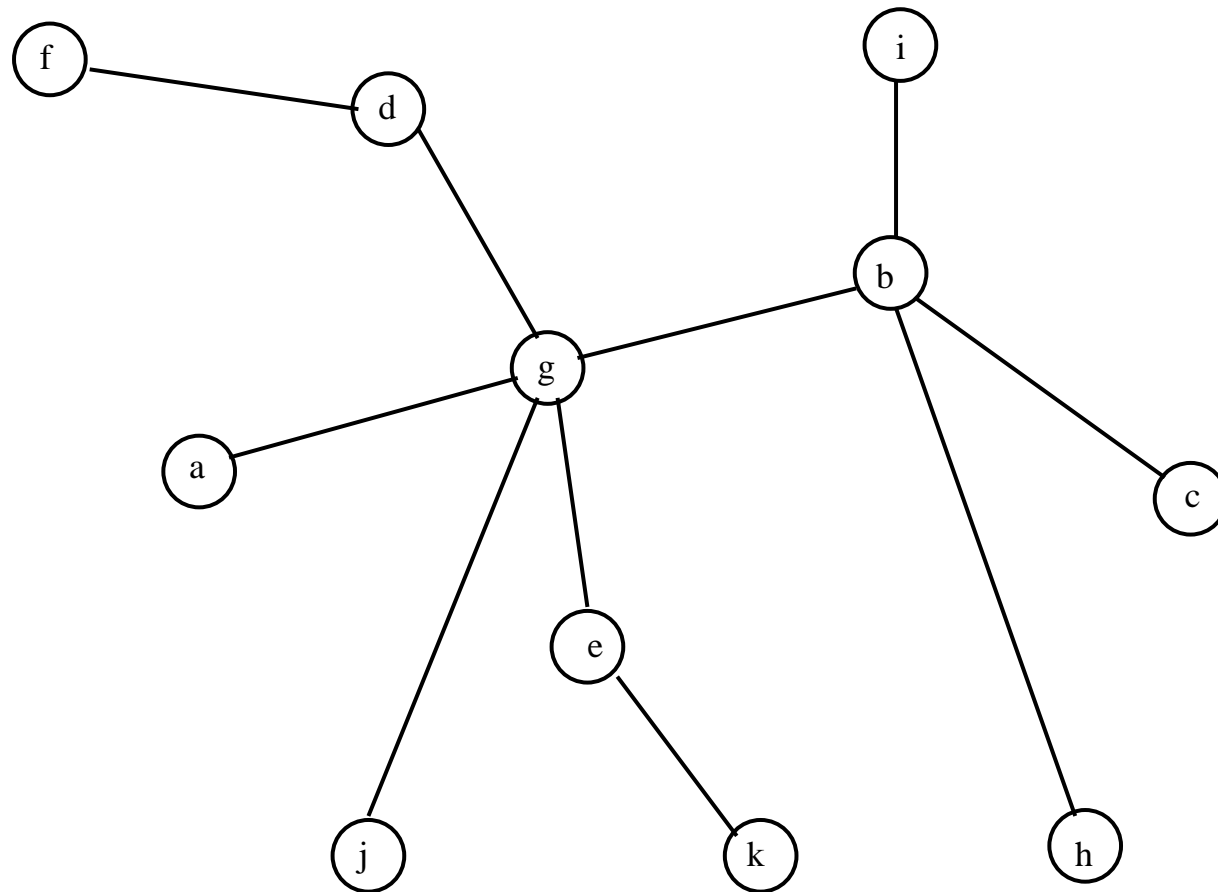
- 5 Usuwamy krawędź e a następnie krawędź c , po czym generujemy kolejno: (a, d) , (a, d, e) . Mamy piątę już drzewo.
- 6 Usuwamy krawędź e , potem d tworzymy nową listę: (a, e) . Nie możemy jednak kontynuować więc usuwamy krawędzie e oraz a . Kolejnymi utworzonymi listami są więc: (b) , (b, c) , (b, c, d) . Otrzymaliśmy drzewo rozpięte.
- 7 Usuwamy krawędź d ale kolejna lista (b, c, e) zawiera cykl. Usuwamy więc kolejno krawędzie e oraz c i generujemy listy: (b, d) , (b, d, e) . Mamy kolejne drzewo rozpięte.

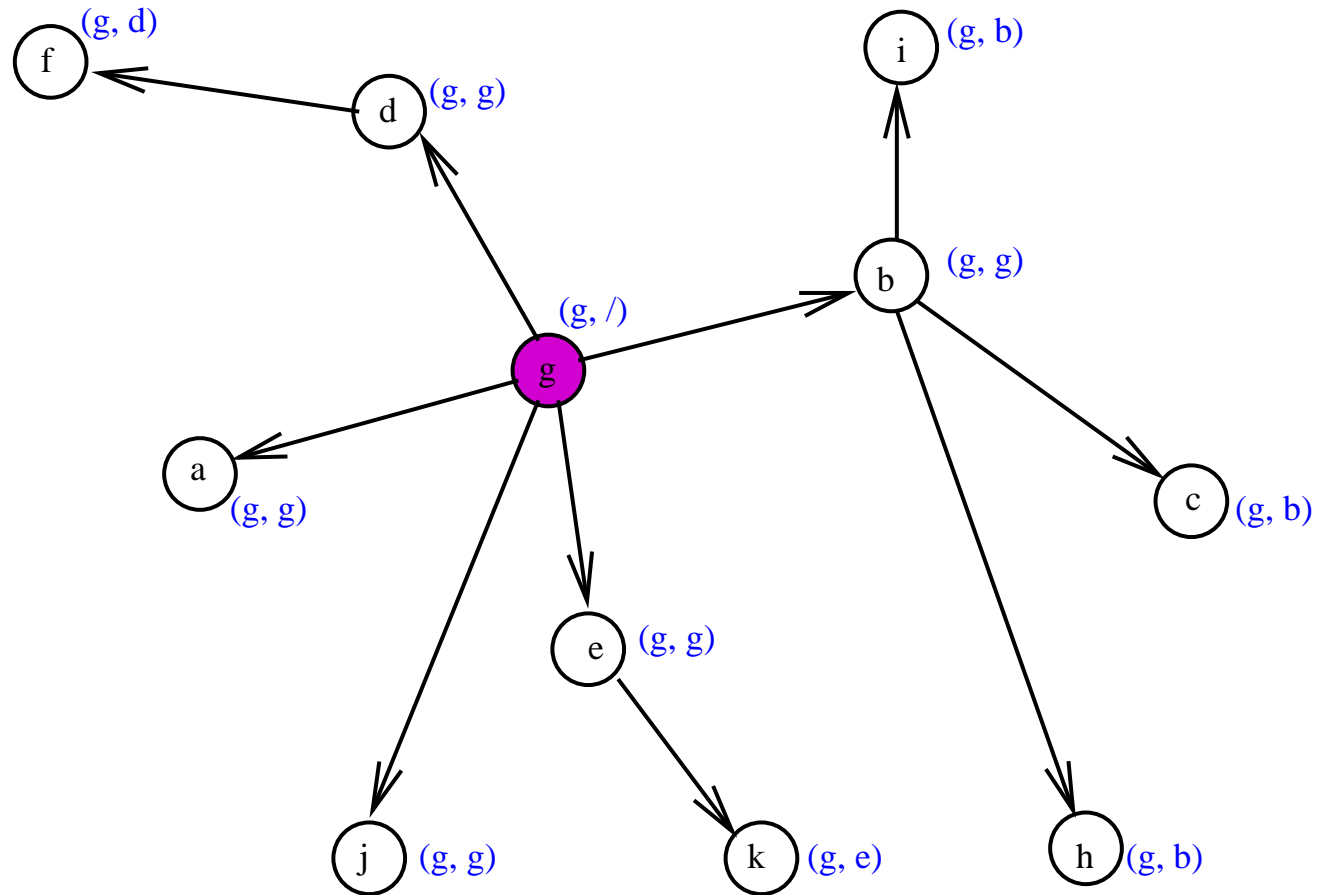
8 Usuwamy kolejno krawędzie e i d a kolejną listą jest: (b, e) . Usuwamy e a następnie b i tworzymy listy: (c) , (c, d) , (c, d, e) . Mamy kolejne drzewo rozpięte.

9 Usuwamy kolejno e i d a następnie generujemy las (c, e) . Usuwamy e , następnie c i generujemy kolejne listy: (d) , (d, e) , (e) . Nie ma już jednak więcej drzew rozpiętych.



- należy znaleźć szybki sposób decydowania czy dodanie krawędzi nie spowoduje powstania cyklu
- w tym celu w każdym drzewie wyróżnimy jeden wierzchołek, zwany *korzeniem*
- każdemu wierzchołkowi v grafu przyporządkujemy dwa atrybuty $Korzeń[v]$, będący korzeniem drzewa w którym znajduje się wierzchołek v oraz $Poprzednik[v]$, będący poprzednikiem wierzchołka v na jedynej ścieżce łączącej v z korzeniem
- poprzednik korzenia jest nieokreślony (w algorytmach zakładamy, że jest równy 0 lub NIL)





- proces generowania wszystkich rozpiętych drzew wymaga dwóch operacji:
 - jeżeli kolejna krawędź z listy nie zamyka cyklu (jest zaakceptowana), to dodanie jej powoduje połączenie dwóch drzew T_1 i T_2 lasu w jedno nowe drzewo T i w związku z tym należy dokonać odpowiedniej zmiany etykiet wierzchołków T (**Procedura A**)
 - po otrzymaniu drzewa rozpiętego, lub wyczerpaniu się naszej listy krawędzi w procesie generowania drzew, wykonujemy krok "powrotu" polegający na wyrzuceniu (ostatnio dodanej) krawędzi, co powoduje rozbięcie pewnego (jednego) drzewa T na dwa poddrzewa T_1 i T_2 (**Procedura B**)

Procedura A

Opis podprocedury zamiany etykiet wierzchołków drzewa T o korzeniu r , po operacji zamiany r na nowy korzeń v .

- wierzchołkowi v jako drugą etykietę (poprzednik) przypisujemy 0 – v staje się korzeniem drzewa T .
- zmieniamy skierowanie (na przeciwne) łuków na ścieżce z r do v i odpowiednio zmieniamy drugie etykiety wierzchołków ($\neq v$) tej ścieżki (drugie etykiety pozostałych wierzchołków w drzewie T pozostają bez zmian)
- wszystkie wierzchołki drzewa T otrzymują pierwszą etykietę (określającą korzeń) równą v .

Algorithm 0.0.1: NOWYKORZEŃ(v)

external $G, \text{Poprzednik}, \text{Korzeń}$

$\text{StaryKorzeń} \leftarrow \text{Korzeń}[v]$

if $\text{StaryKorzeń} = v$ **then exit**

$v_1 \leftarrow 0; v_2 \leftarrow v$

repeat

$p \leftarrow v_1; v_1 \leftarrow v_2$

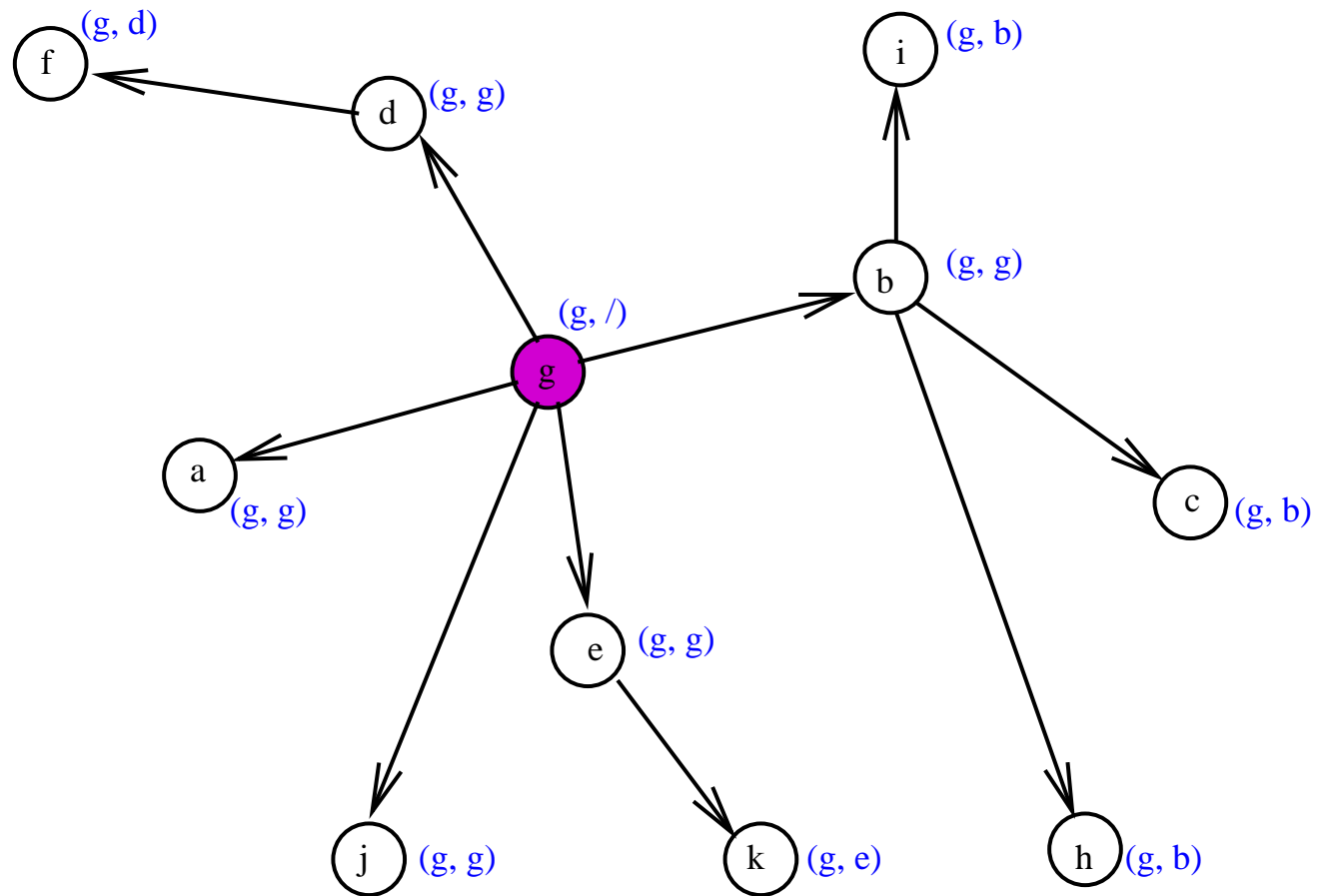
$v_2 \leftarrow \text{Poprzednik}[v_1]$

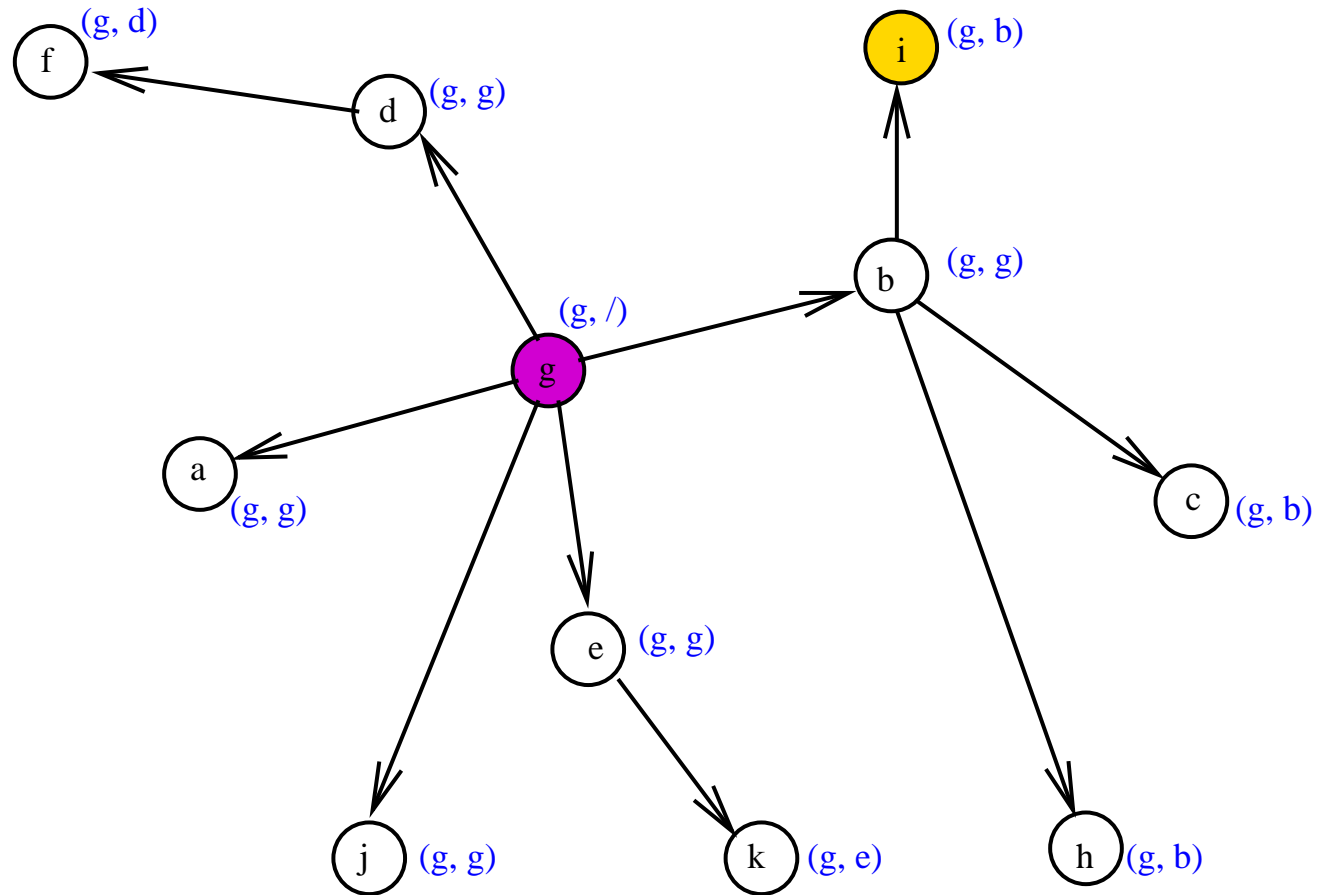
$\text{Poprzednik}[v_1] \leftarrow p$

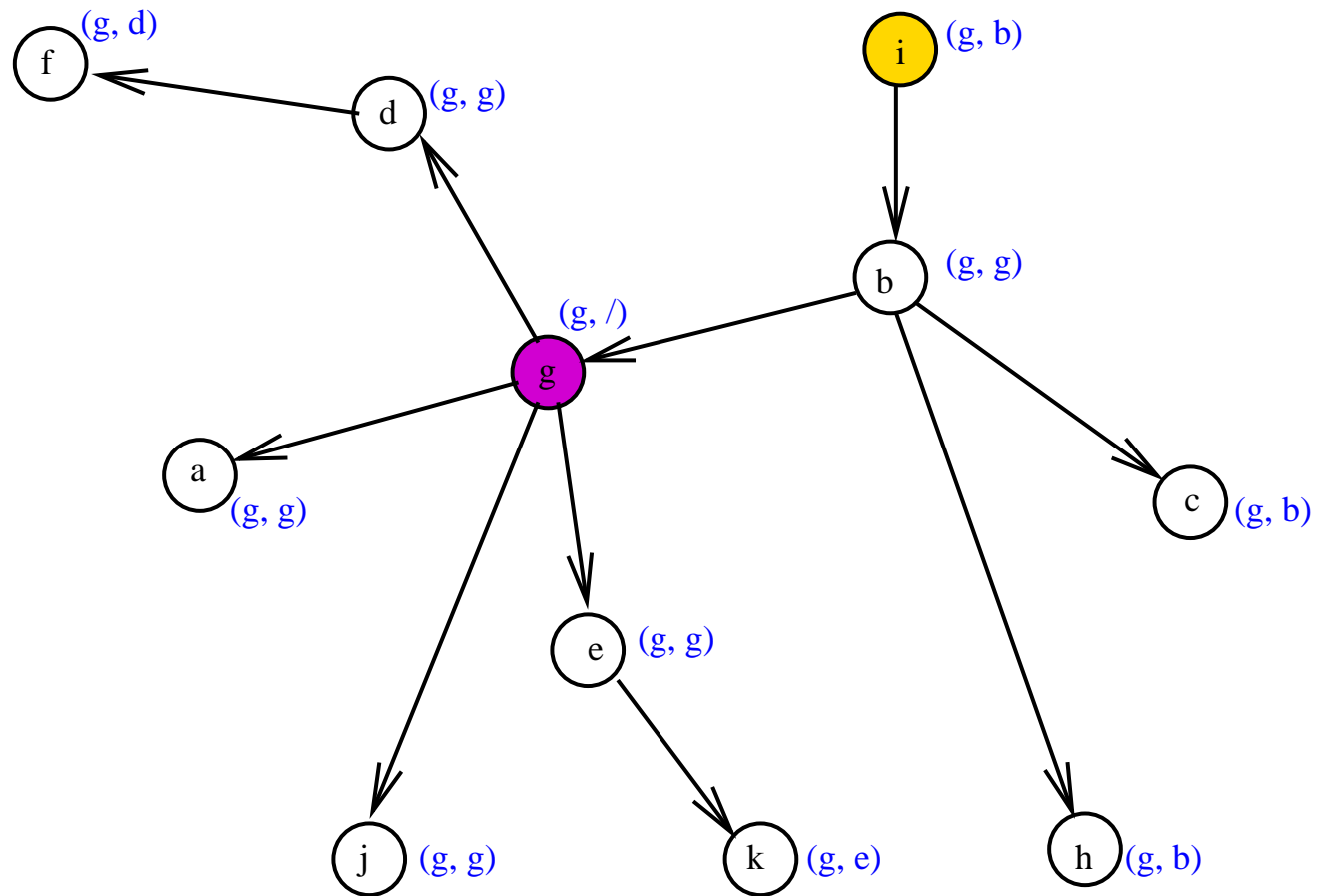
until $v_1 = \text{StaryKorzeń}$

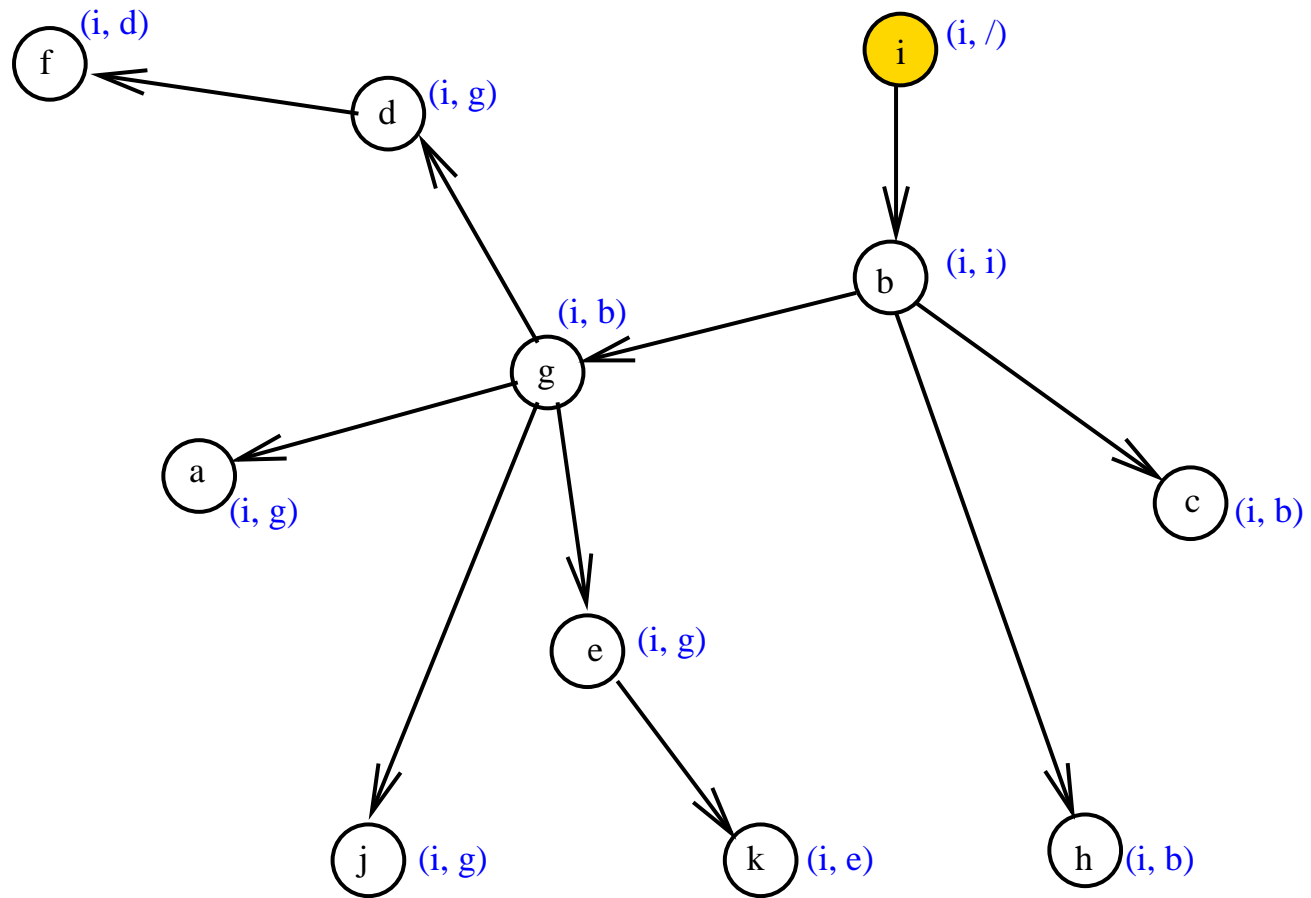
for each $w \in V(G)$

do if $\text{Korzeń}[w] = \text{StaryKorzeń}$ **then** $\text{Korzeń}[w] \leftarrow v$









Opis podprocedury zamiany etykiet wierzchołków dwóch drzew T_1 i T_2 , o korzeniach odpowiednio r_1 i r_2 , $r_1 < r_2$, po operacji ich połączenia przez dodanie krawędzi $e = (u, v)$, gdzie $u \in V(T_1)$, $v \in V(T_2)$.

- wierzchołkom w drzewie T_2 zmieniamy drugie etykiety dotyczące poprzedników tak jak przy zamianie korzenia z r_1 na v (patrz procedura powyżej); w drzewie T_1 drugie etykiety wierzchołków pozostają bez zmian
- wierzchołkowi v jako drugą etykietę (poprzednik) przypisujemy u .
- wszystkie wierzchołki drzewa T_2 otrzymują pierwszą etykietę (określającą korzeń) równą r_1 ; w drzewie T_1 pierwsze etykiety wierzchołków pozostają bez zmian

Algorithm 0.0.1: DODAJKRAWĘDŹ($e = (u, v)$)

external $G, \text{Poprzednik}, \text{Korzeń}$

$v_1 \leftarrow \text{Korzeń}[u]; v_2 \leftarrow \text{Korzeń}[v]$

if $v_2 < v_1$

then $v_1 \leftrightarrow v_2; u \leftrightarrow v$

$\text{NowyKorzeń}(v)$

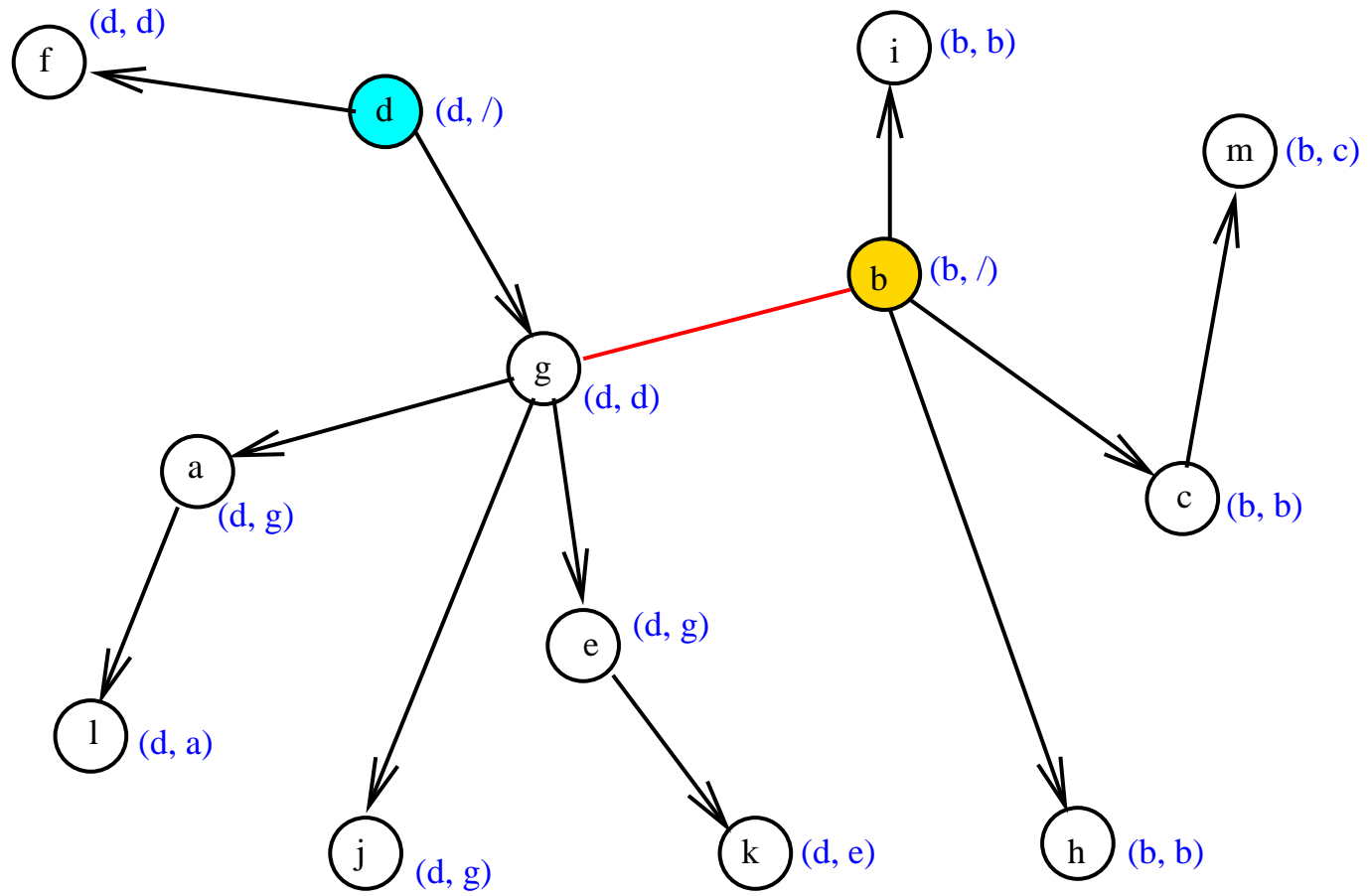
$\text{Poprzednik}[v] \leftarrow u$

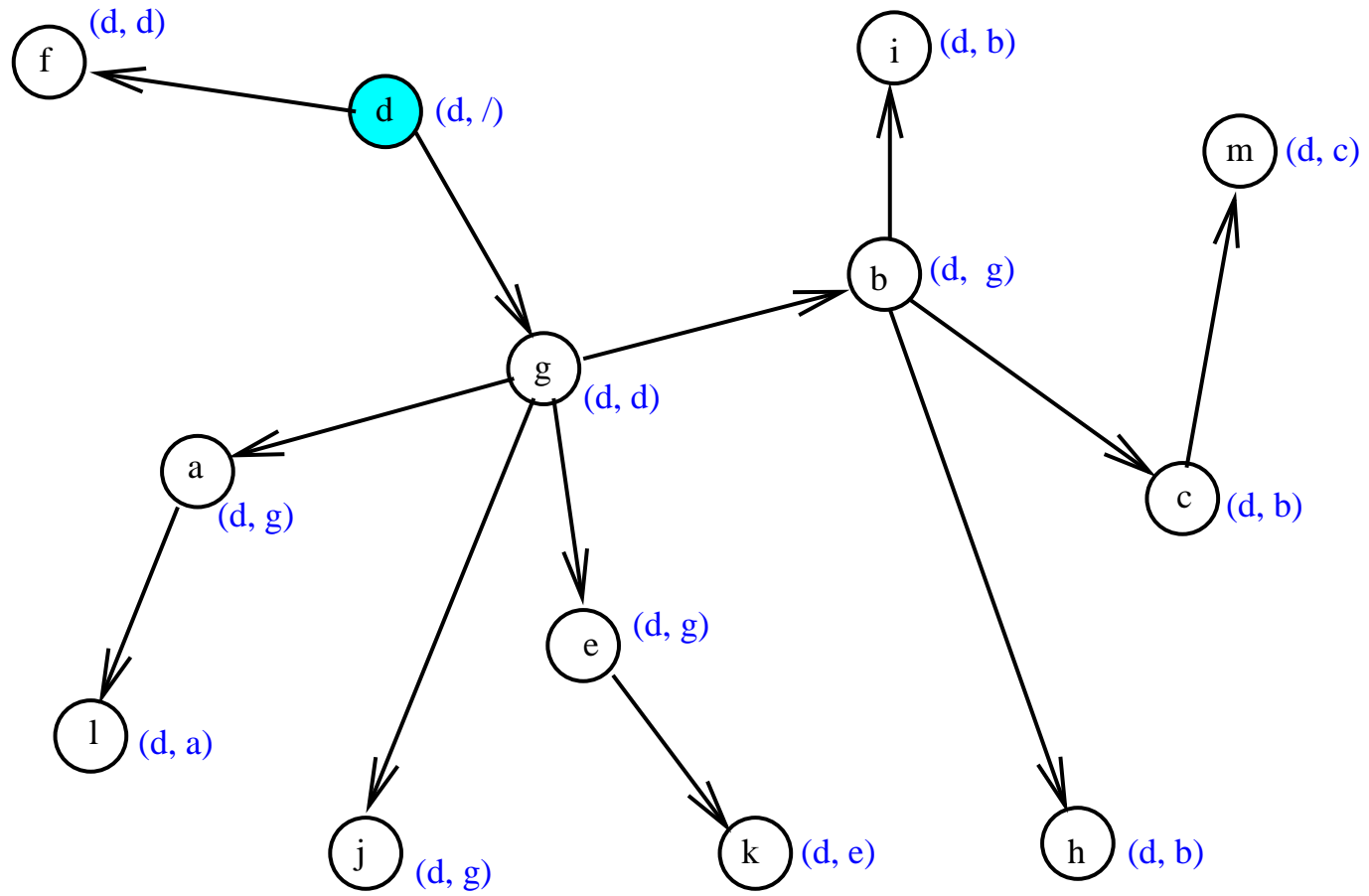
for each $w \in V(G)$

do if $\text{Korzeń}[w] = v_2$ **then** $\text{Korzeń}[w] \leftarrow v_1$









Procedura B

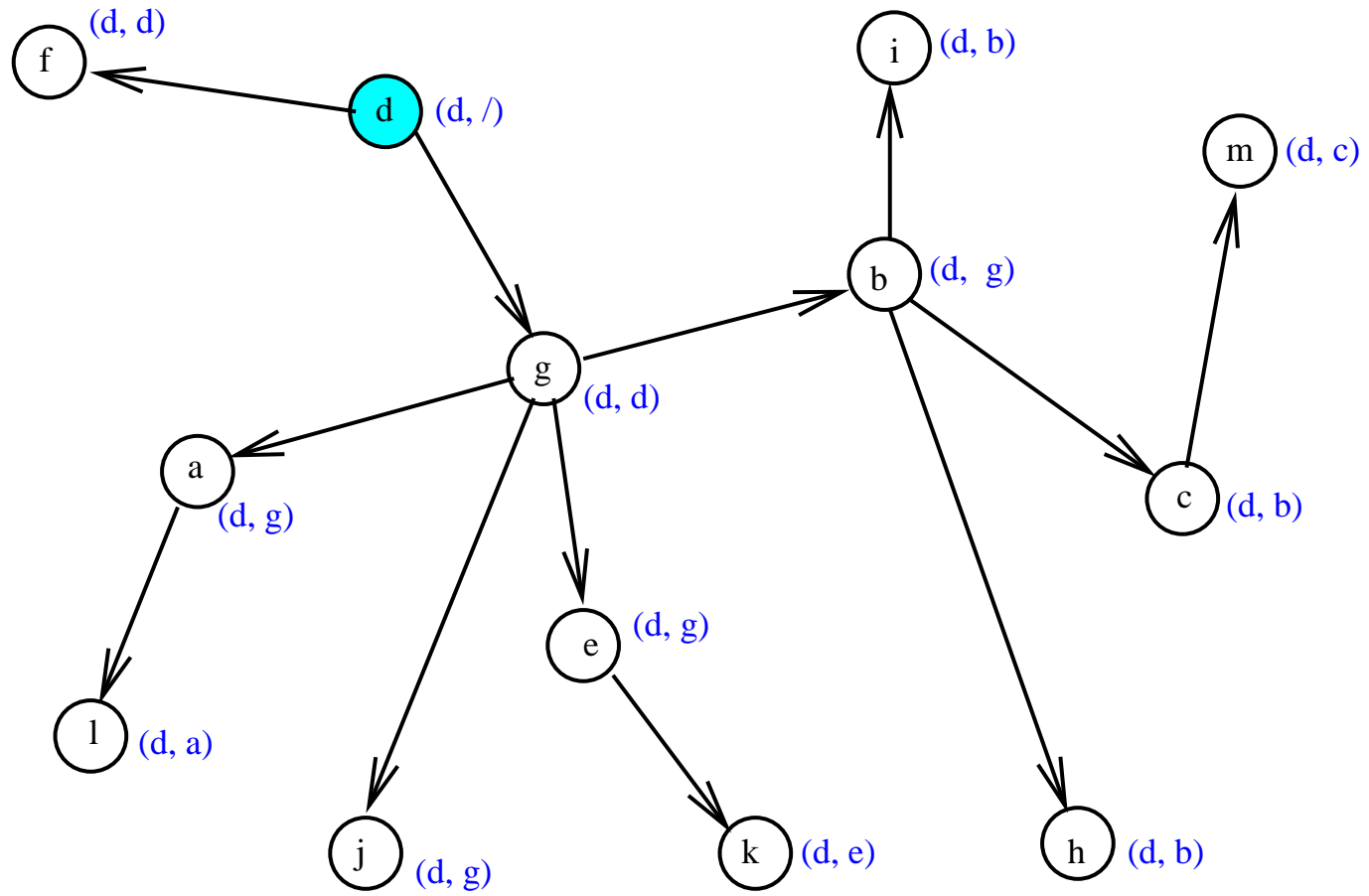
Opis procedury zamiany etykiet wierzchołków drzewa T o korzeniu r po usunięciu z niego krawędzi $e = (u, v)$, czyli rozbiciu T na dwa drzewa T_1 i T_2 , gdzie $u \in V(T_1)$, $v \in V(T_2)$ i $r \in V(T_1)$.

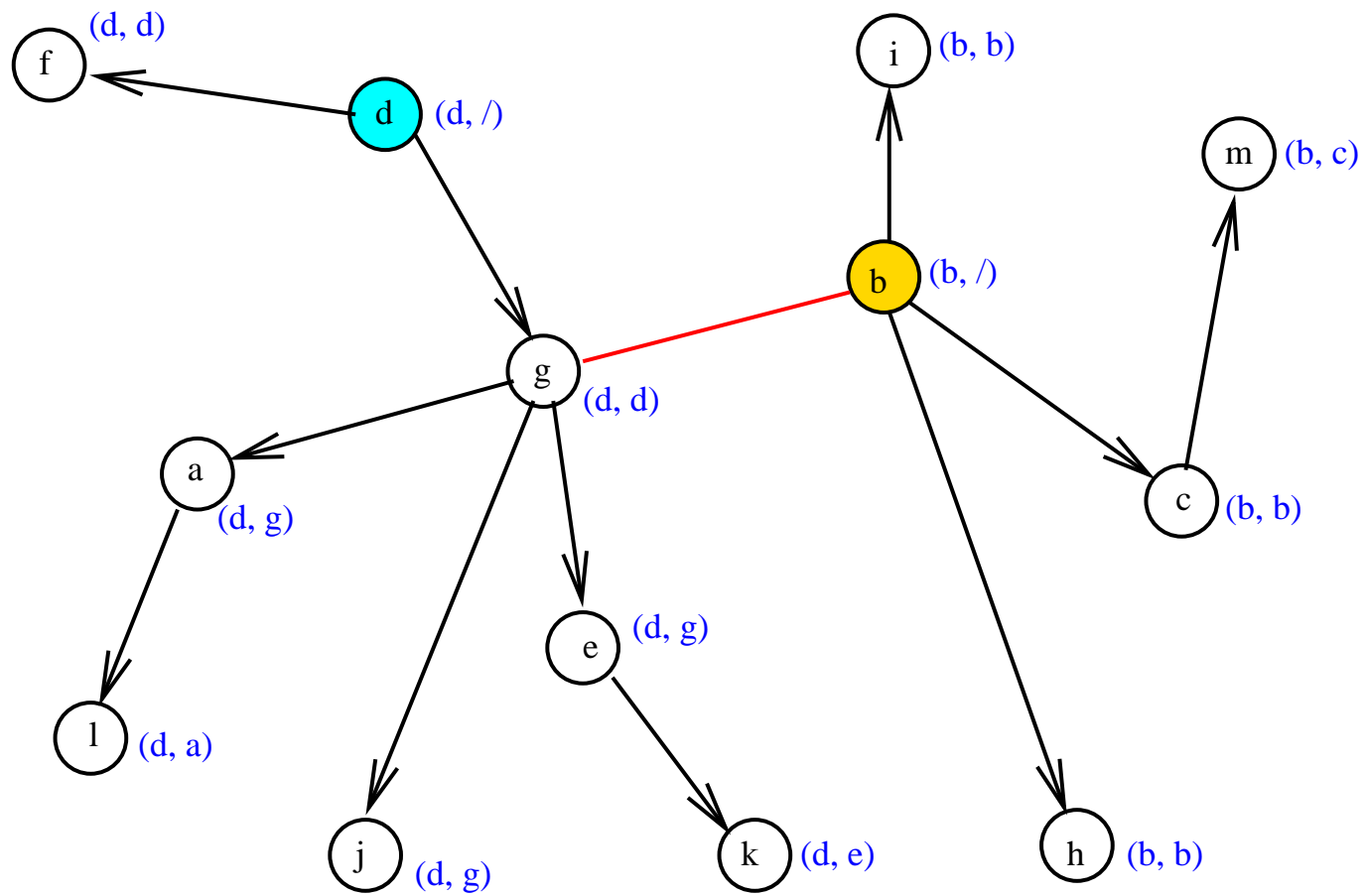
- w drzewie T_1 obie etykiety wierzchołków pozostają bez zmian
- wierzchołkowi v jako drugą etykietę (poprzednik) przypisujemy 0 – v staje się korzeniem drzewa T_2 (drugie etykiety pozostałych wierzchołków w drzewie T_2 pozostają bez zmian)
- wszystkie wierzchołki drzewa T_2 otrzymują pierwszą etykietę (określającą korzeń) równą v .

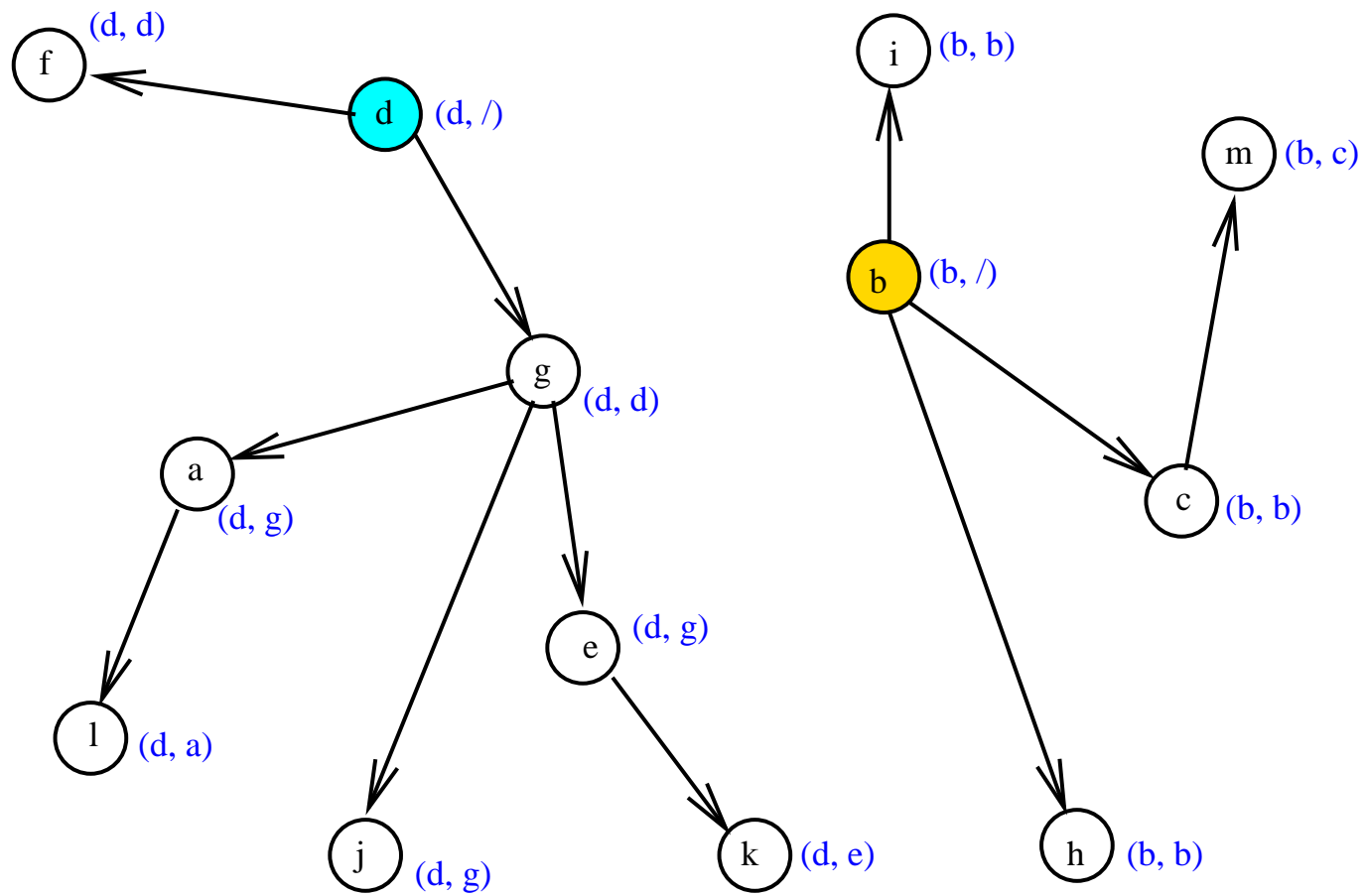
UWAGA! W naszym przypadku zawsze usuwamy ostatnio dodaną krawędź, więc wierzchołek u jest wierzchołkiem wiszącym i po usunięciu krawędzi stanie się drzewem trywialnym (jednowierzchołkowym). Zatem proces usunięcia krawędzi polega jedynie na zmianie atrybutów wierzchołka u . Realizuje to algorytm USUŃKRAWĘDŹ.

Algorithm 0.0.1: USUŃKRAWĘDŹ($e = (u, v)$)

```
external  $G, \text{Poprzednik}, \text{Korzeń}$   
if  $\text{Poprzednik}[u] = v$  then  $u \leftrightarrow v$   
 $\text{Poprzednik}[v] \leftarrow 0$   
 $\text{Korzeń}[v] \leftarrow v$ 
```







Algorytm generowania wszystkich drzew rozpiętych grafu

- 1 Utwórz listę krawędzi biorąc na jej początek krawędzie incydentne z dowolnie wybranym wierzchołkiem v^* :

$$e_1, e_2, \dots, e_{d(v^*)}, e_{d(v^*)+1}, \dots, e_m.$$

Dla każdego wierzchołka $v \in V(G)$:

$$\textit{Korzeń}[v] \leftarrow v; \quad \textit{Poprzednik}[v] \leftarrow 0.$$

Nadajemy wartości początkowe zmiennym: $k \leftarrow 1$ (k – wskaźnik na listę krawędzi), $\textit{koniec} \leftarrow d(v^*) + 1$ (możemy także nadać wartość $\textit{koniec} \leftarrow m$, wówczas wygenerujemy również wszystkie lasy rozpięte).

- 2 Jeżeli $k = m + 1$, to przejdź do kroku 5. W przeciwnym razie badamy k -tą krawędź z listy: $e_k = (u_k, v_k)$.
- Jeżeli $Korzeń(u_k) = Korzeń(v_k)$, to u_k i v_k są w tym samym drzewie i krawędź e_k zamyka cykl – odrzucamy ją; $k \leftarrow k + 1$; przejdź do kroku 2;
 - Jeżeli $Korzeń(u_k) \neq Korzeń(v_k)$, to krawędź akceptuj i przejdź do kroku 3;
- 3 Połącz dwa drzewa krawędzią $e_k = (u_k, v_k)$ zgodnie z procedurą DODAJKRAWĘDŹ($e_k = (u_k, v_k)$);

- 4 Jeżeli liczba zaakceptowanych krawędzi jest równa $n - 1$, to mamy drzewo rozpięte. Zapamiętaj je i przejdź do kroku 5, jeżeli krawędzi zaakceptowanych jest mniej, to $k \leftarrow k + 1$ i przejdź do kroku 2;
- 5 Badamy ostatnio zaakceptowaną krawędź. Powiedzmy, że jest nią krawędź e_l . Jeżeli $e_l = e_{koniec}$ i nie ma już innych zaakceptowanych krawędzi, to STOP (mamy wyznaczone wszystkie rozpięte drzewa). W przeciwnym razie odrzuć krawędź e_l i zmień etykiety zgodnie z procedurą $USUŃKRAWĘDŹ(e_l = (u_l, v_l))$; $k \leftarrow k + 1$ i przejdź do kroku 2.

Algorithm 0.0.1: WSZYSTKIEDRZEWA($G, koniec$)

comment: Zakładamy, że $e_k = u_k v_k$

external NOWYKORZEŃ(), DODAJKRAWĘDŹ(), USUŃKRAWĘDŹ()

for each $v \in V(G)$

do $Korzeń[v] = v$; $Poprzednik[v] \leftarrow 0$

$k \leftarrow 1$; $i \leftarrow 0$

repeat

if $Korzeń[u_k] \neq Korzeń[v_k]$

then $\begin{cases} \text{DODAJKRAWĘDŹ}(e_k) \\ i \leftarrow i + 1 \\ Drzewo[i] \leftarrow e_k \end{cases}$

if $i = \nu(G) - 1$ **then output** ($Drzewo$)

if $i = \nu(G) - 1 \vee k = \varepsilon(G)$

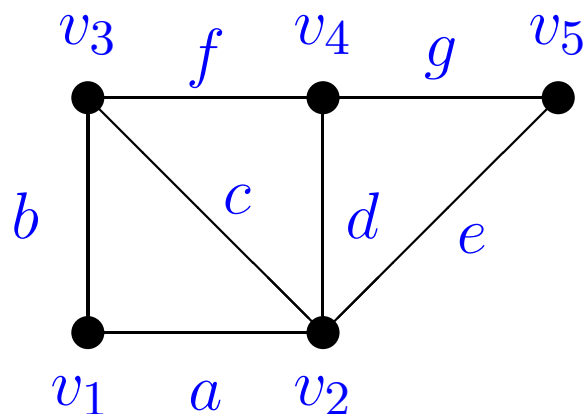
then $\begin{cases} \text{USUŃKRAWĘDŹ}(Drzewo[i]) \\ k \leftarrow Drzewo[i] + 1 \\ i \leftarrow i - 1 \end{cases}$

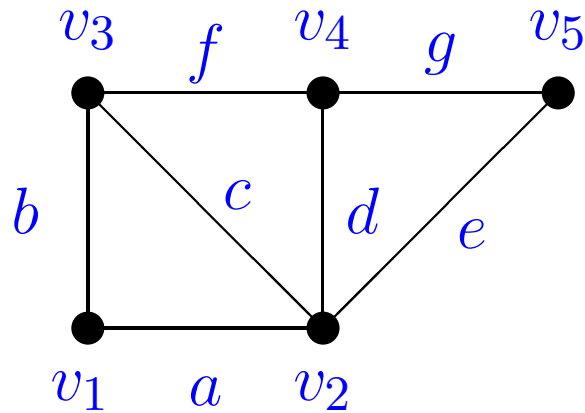
else $k \leftarrow k + 1$

until $Drzewo[1] = e_{koniec}$

Przykład . Znaleźć wszystkie drzewa rozpięte grafu G o macierzy incydencji:

$$M(G) = \begin{matrix} & \begin{matrix} a & b & c & d & e & f & g \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} & \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{matrix} .$$





$-$, a , ab , ~~abc~~ , abd , $abde$, ~~$abdf$~~ , $abdg$, $abef$, $abeg$, $abfg$,
 abg , ac , acd , $acde$, ~~$acdf$~~ , $acd g$, ace , $acef$, $aceg$, acf , $acfg$,
 acg , ad , ade , $adef$, ~~$adeg$~~ , adf , $adfg$, adg , $ae f$, $ae fg$, aeg ,
 af , afg , ag ,
 b , bc , bcd , $bcde$, ~~$bcd f$~~ , $bcd g$, bce , $bcef$, $bceg$, bcf , $bcfg$, bcg ,
 bd , bde , $bdef$, ~~$bdeg$~~ , $bd f$, $bd fg$, bdg , be , $be f$, $be fg$, beg , bf ,
 $bf g$, bg , c .

6.3 Rozpięte drzewa o minimalnej wadze (MST)

- Jeżeli mamy do czynienia z grafem z wagami, to najczęściej interesuje nas znalezienie *rozpiętego drzewa o minimalnej wadze*, to znaczy drzewa z najmniejszą sumą wag jego krawędzi.
- Aby znaleźć drzewo o żądanych własnościach możemy zastosować dwa algorytmy:
 - Kruskala („algorytm zachłanny”)
 - Prima („algorytm najbliższego sąsiada”)
- Niech graf $G = (V, E)$ będzie dany macierzą wag i założmy, że graf G jest spójny.

Algorytm Kruskala

- 1 Wybierz krawędź, która nie jest pętlą, e_1 tak, by waga tej krawędzi była najmniejsza.
- 2 Jeżeli krawędzie e_1, e_2, \dots, e_k zostały już wybrane, to z pozostałych $E \setminus \{e_1, e_2, \dots, e_k\}$ wybierz krawędź e_{k+1} w taki sposób aby:
 - graf, który składa się tylko z krawędzi $e_1, e_2, \dots, e_k, e_{k+1}$ był **acykliczny**, oraz
 - waga krawędzi e_{k+1} była **najmniejsza**.
- 3 Jeśli nie można wykonać kroku 2, to STOP.

Algorithm 0.0.1: KRUSKAL(G, w)

external DODAJKRAWĘDŹ()

for each $v \in V(G)$

do $Korzeń[v] = v$; $Poprzednik[v] \leftarrow 0$

$Sortuj(E)$

$Drzewo \leftarrow \emptyset$; $k \leftarrow 1$

repeat

$u, v \leftarrow$ końce krawędzi e_k

if $Korzeń[u] \neq Korzeń[v]$

then DODAJKRAWĘDŹ(e); $Drzewo \leftarrow Drzewo \cup \{e_k\}$

$E \leftarrow E \setminus \{e_k\}$; $k \leftarrow k + 1$

until $|Drzewo| = |V| - 1 \vee |E| = 0$

if $|Drzewo| \neq |V| - 1$ **then output** (Graf G nie jest spójny!)

return ($Drzewo$)

Złożoność obliczeniowa algorytmu Kruskala.

Algorytm można podzielić na dwa etapy:

- w pierwszym etapie sortujemy krawędzie według wag w czasie $O(m \log m)$
- w drugim etapie budujemy rozpięte drzewo poprzez wybór najkrótszych krawędzi ze zbioru krawędzi $E(G)$; ten etap można wykonać w czasie $O(m \log n)$
- sumaryczny czas pracy algorytmu Kruskala wynosi:

$$O(m \log n)$$

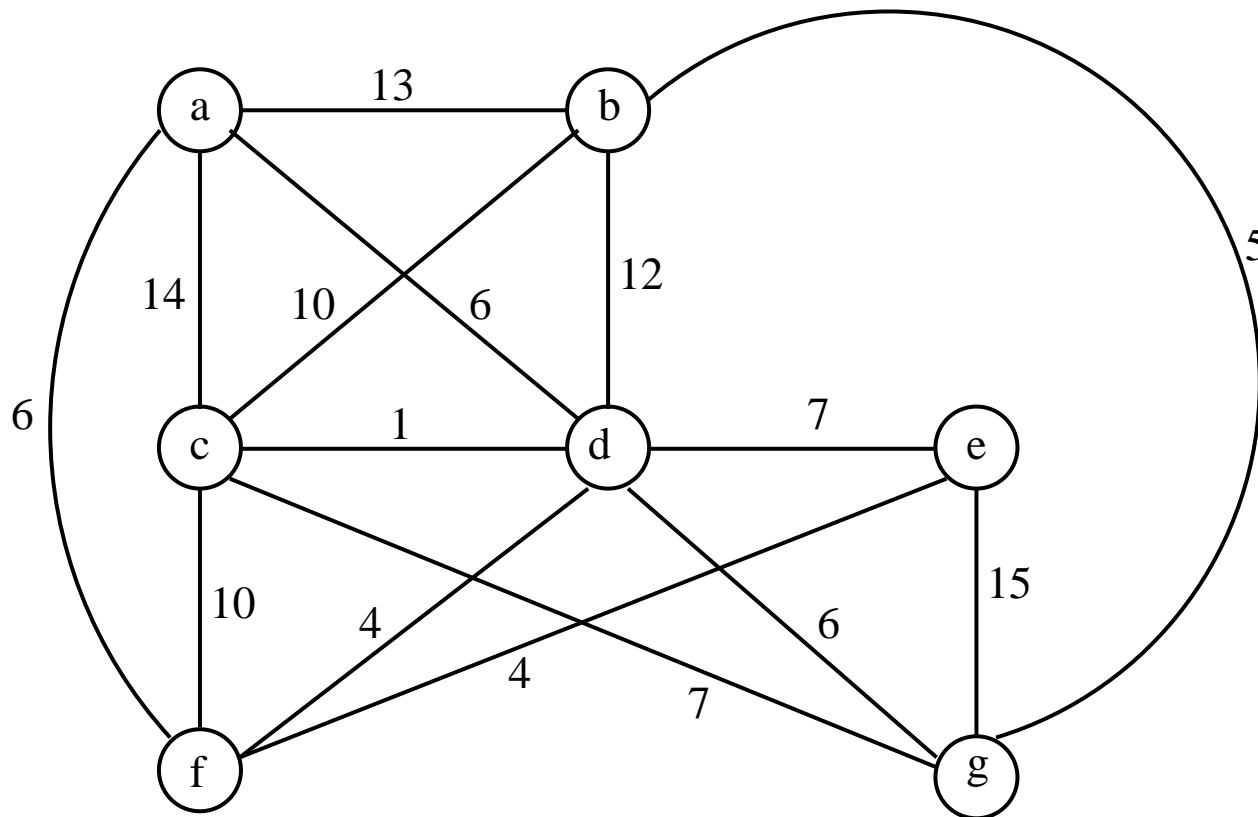
Twierdzenie . *Drzewo rozpięte skonstruowane za pomocą algorytmu Kruskala jest optymalne.*

Dowód (nie wprost)

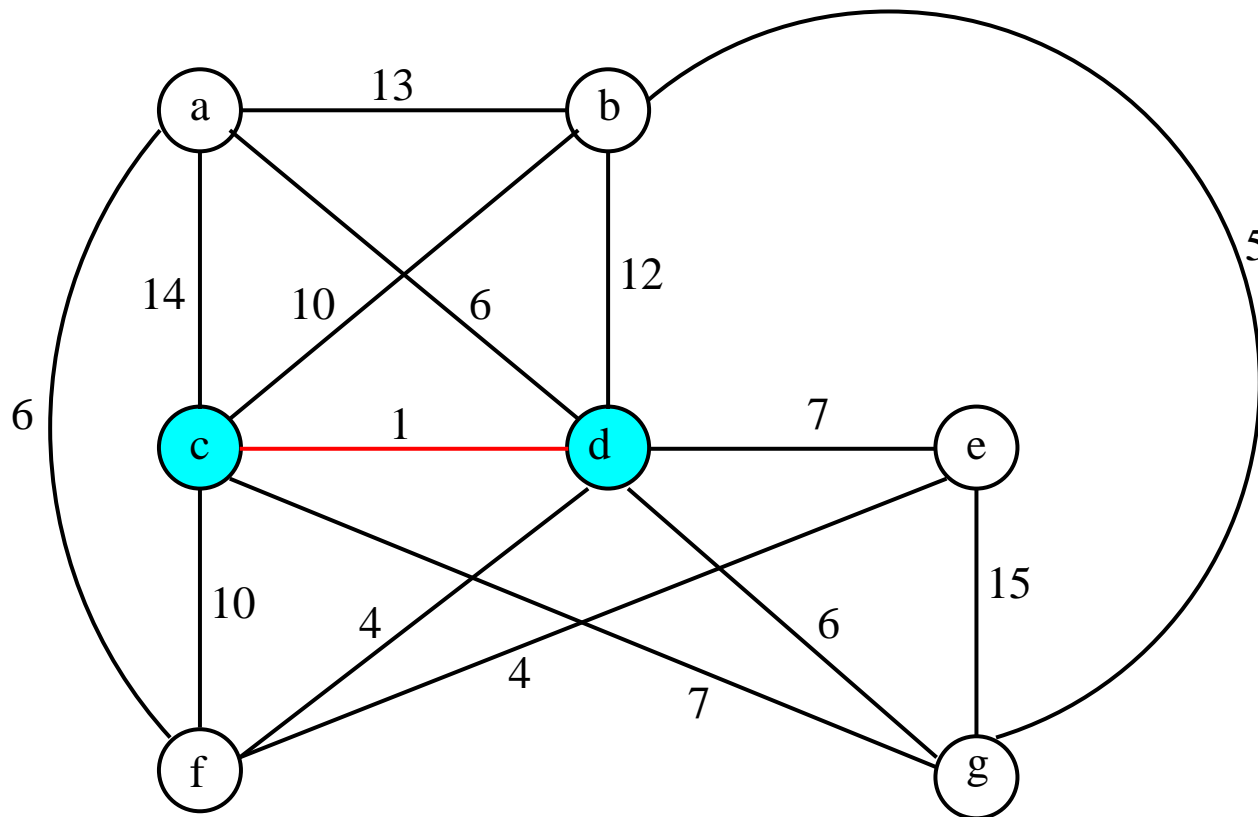
- Przypuśćmy, że drzewo \tilde{T} skonstruowane przez algorytm Kruskala nie jest optymalne (minimalne).
- Weźmy optymalne drzewo rozpięte T danego grafu, takie, że T i \tilde{T} mają wspólne krawędzie e_1, \dots, e_{k-1} dla możliwie największego k .
- Dodanie krawędzi e_k drzewa \tilde{T} do drzewa T (z definicji $k, e_k \notin T$) spowoduje powstanie dokładnie jednego cyklu. Weźmy krawędź e należącą do tego cyklu, $e \neq e_k$ taką, która nie należy do \tilde{T} .

- Z tego, że algorytm Kruskala dodaje krawędzie o najmniejszej wadze wynika, że $w(e) \geq w(e_k)$.
- Gdyby $w(e) > w(e_k)$, to podgraf $T' = (T + e_k) - e$ byłby nie tylko acykliczny i spójny (czyli byłby drzewem), ale również byłoby to drzewo rozpięte z wagą mniejszą niż waga T , co przeczy założeniu, że T jest optymalne.
- Wobec tego $w(e) = w(e_k)$, zatem T' jest również optymalne, co przeczy założeniu, że k jest możliwie największe.
- A więc założenie, że \tilde{T} nie jest optymalne doprowadziło do sprzeczności. Wynika więc stąd, że drzewo \tilde{T} otrzymane za pomocą algorytmu Kruskala jest optymalne, co kończy dowód.

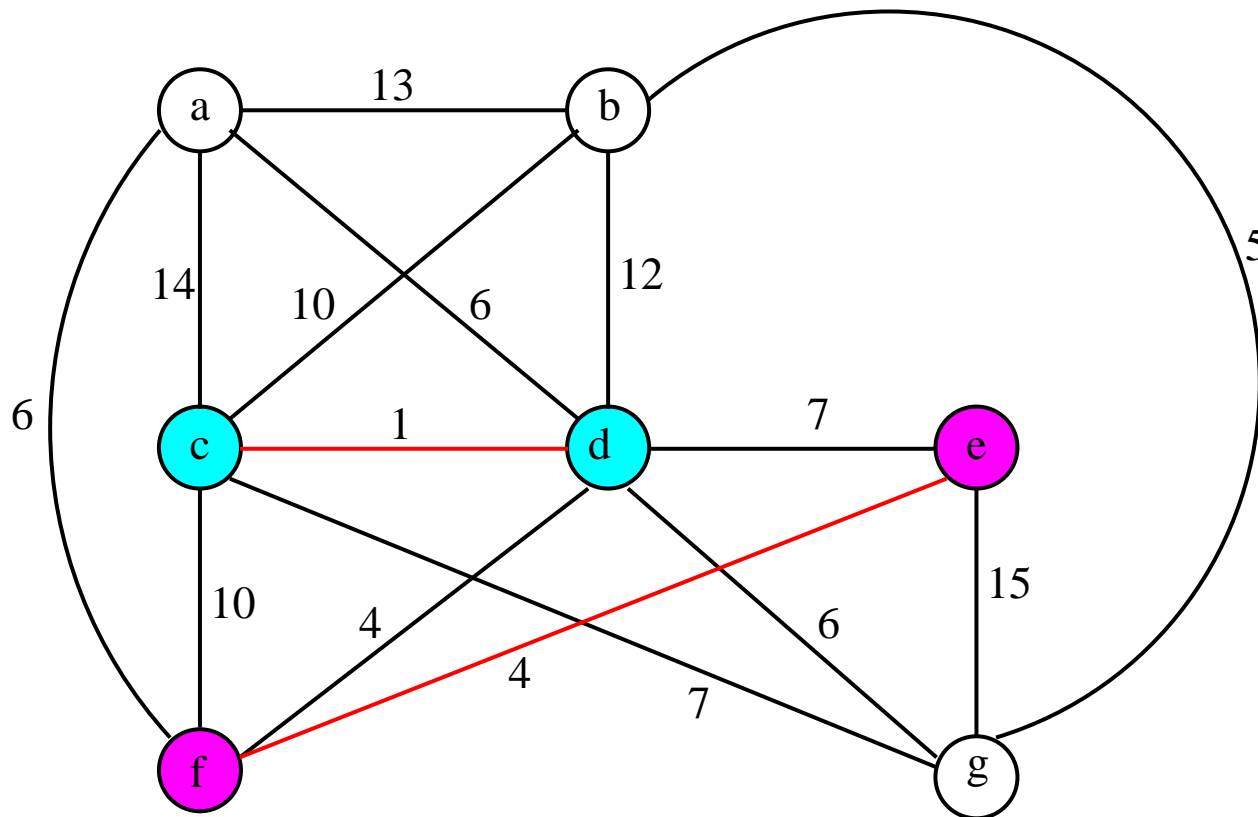
Przykład – algorytm Kruskala



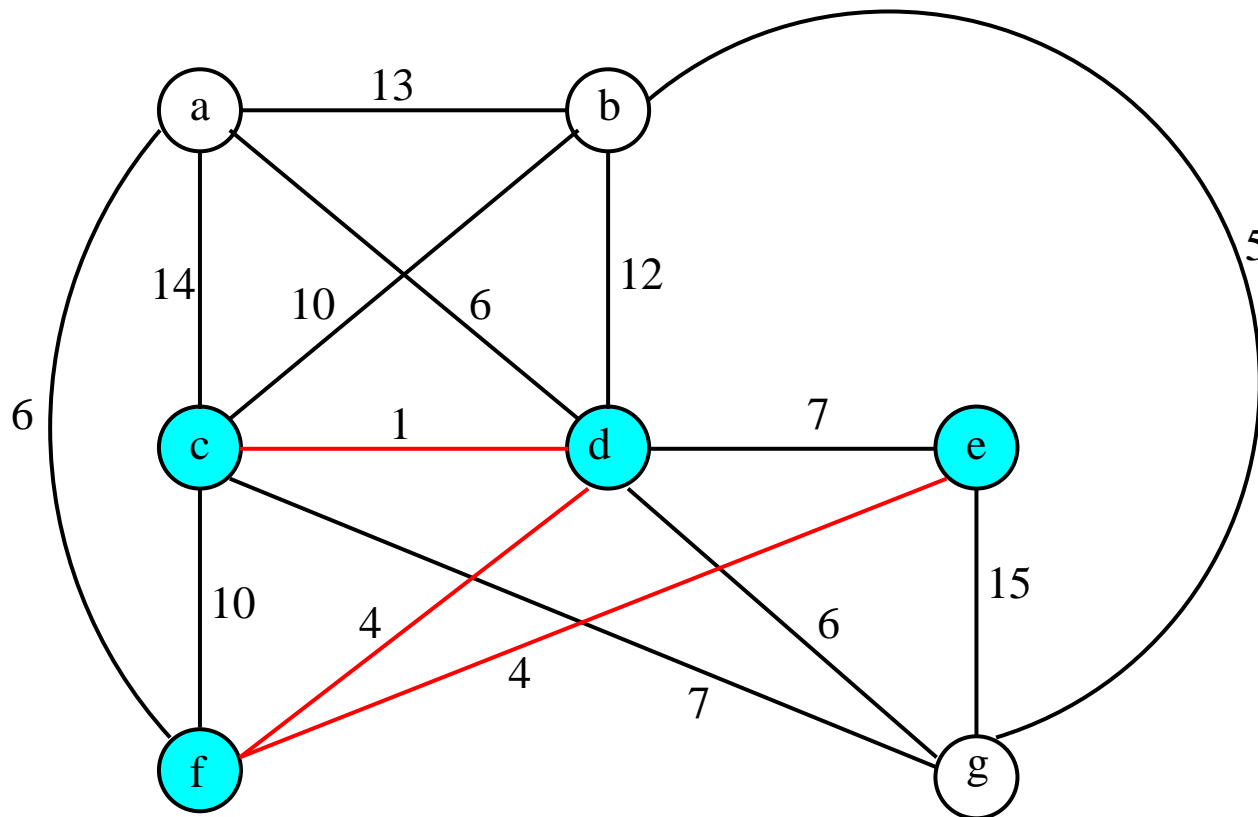
Przykład – algorytm Kruskala



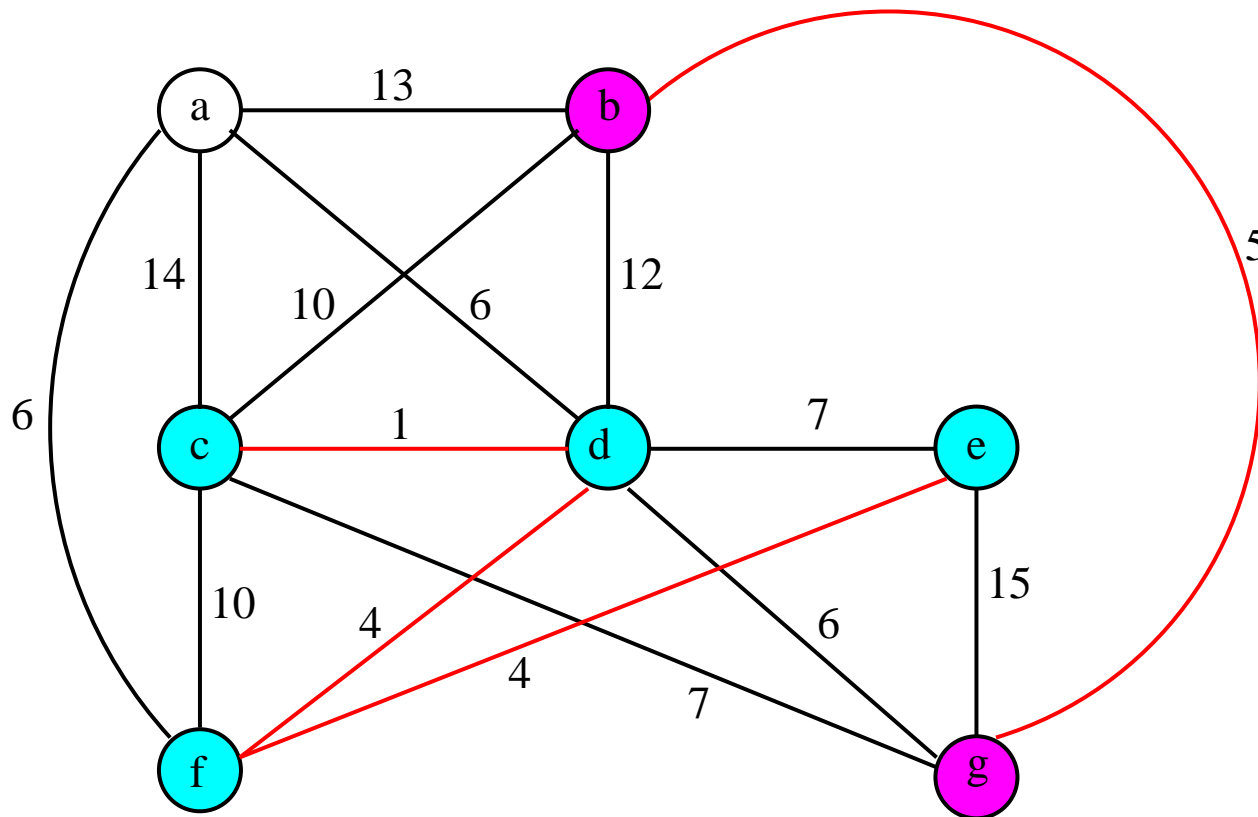
Przykład – algorytm Kruskala



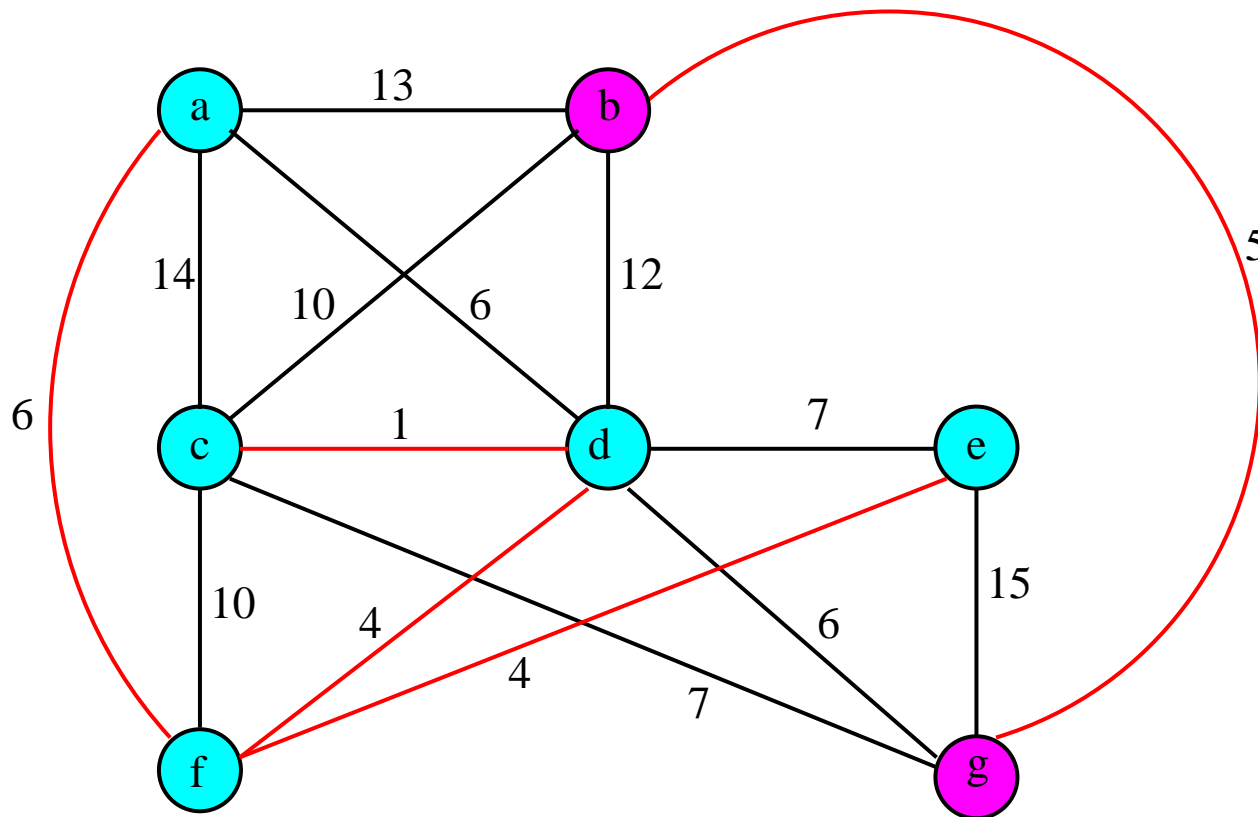
Przykład – algorytm Kruskala



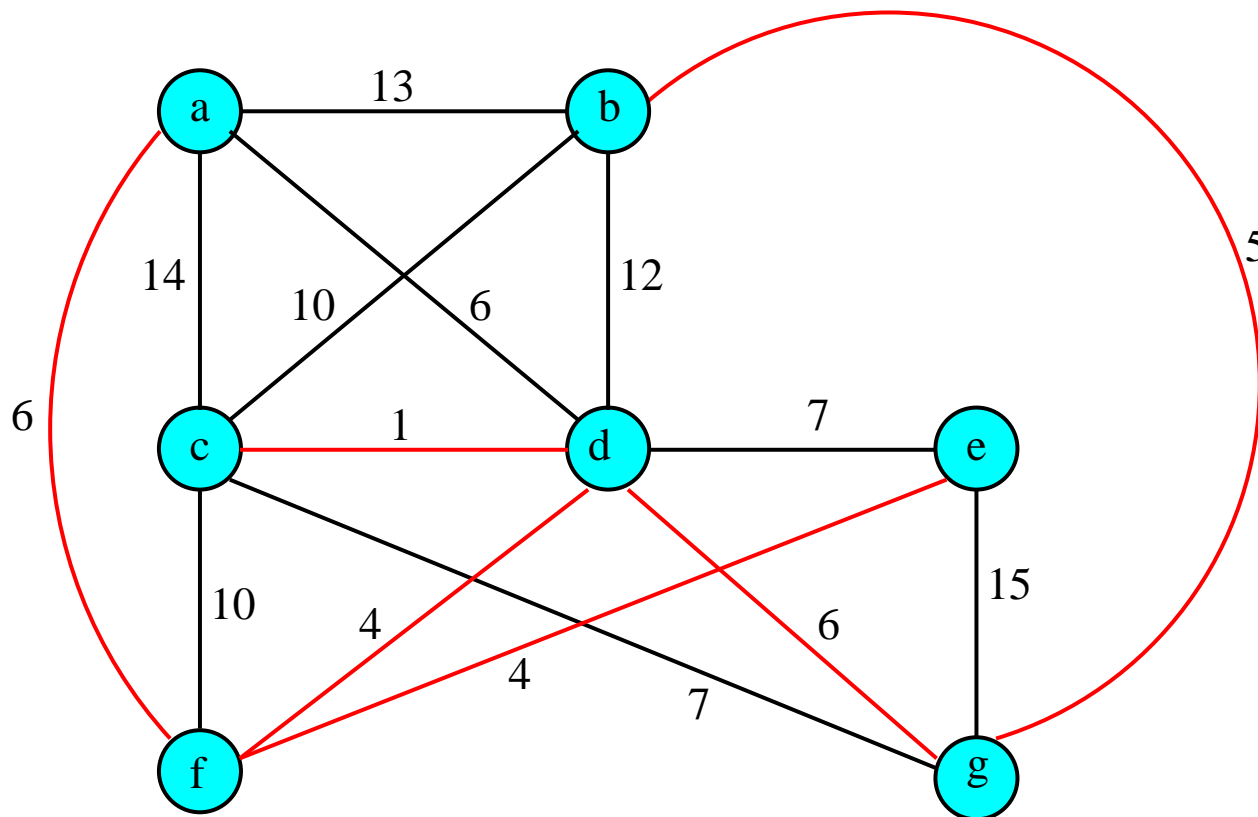
Przykład – algorytm Kruskala



Przykład – algorytm Kruskala



Przykład – algorytm Kruskala



Algorytm Prima

- Zaczynamy od dowolnego wierzchołka u w danym grafie. Niech uv będzie krawędzią o najmniejszej wadze incydentną z u . Krawędź ab włączamy do drzewa.
- Następnie spośród wszystkich krawędzi incydentnych albo do u albo do v wybieramy krawędź o najmniejszej wadze i włączamy do częściowo zbudowanego drzewa. W wyniku tego w skład wierzchołków drzewa wszedł nowy wierzchołek, powiedzmy z .

- Powtarzając opisany proces szukamy krawędzi o najmniejszej wadze łączącej wierzchołki u , v lub z z innym wierzchołkiem grafu. Kontynuujemy to postępowanie dopóki wszystkie wierzchołki grafu G nie znajdą się w drzewie, czyli dopóki drzewo nie „stanie się” rozpięte.

W poniższym algorytmie każdemu wierzchołkowi v_i będziemy przypisywać parę etykiet (α_i, β_i) , gdzie α_i jest wierzchołkiem poddrzewa najbliższym v_i , zaś β_i jest bieżącą odległością wierzchołka v_i od poddrzewa, czyli wagą krawędzi (v_i, α_i) .

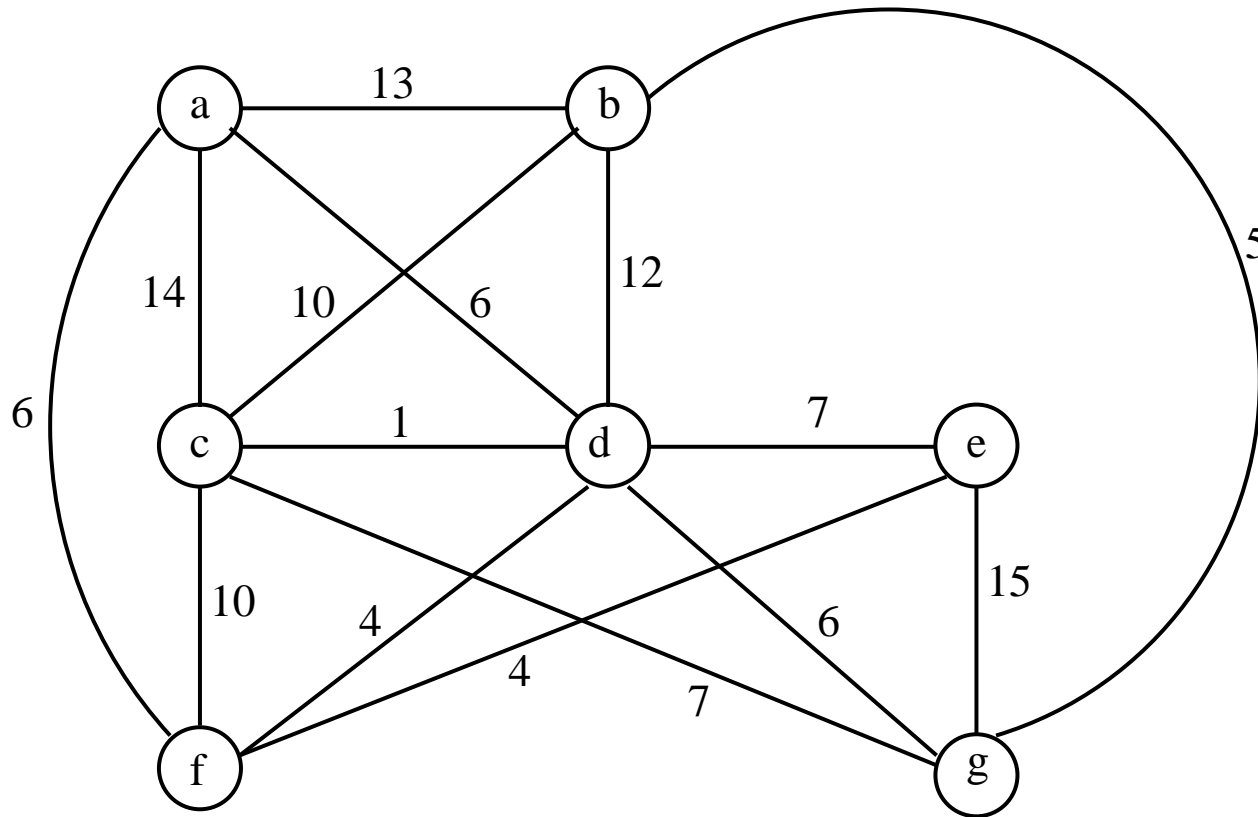
Algorithm 0.0.1: PRIM(G, W, v_0)

```
for each  $v \in V(G)$ 
  do  $\begin{cases} \beta[v] \leftarrow W[v, v_0] \\ \textbf{if } vv_0 \in E(G) \textbf{ then } \alpha[v] \leftarrow v_0 \end{cases}$ 
 $T \leftarrow \{v_0\}; \textit{Drzewo} \leftarrow \emptyset; k \leftarrow 1$ 
repeat
   $w_{min} \leftarrow \infty$ 
  for each  $v \in V(G) \setminus T$ 
    do if  $\beta[v] < w_{min}$  then  $w_{min} \leftarrow \beta[v]; v_{min} \leftarrow v$ 
  if  $w_{min} = \infty$  then output (Graf  $G$  nie jest spójny!)
   $T \leftarrow T \cup \{v_{min}\}$ 
   $\textit{Drzewo} \leftarrow \textit{Drzewo} \cup \{\text{krawędź łącząca } v_{min} \text{ i } \alpha v_{min}\}$ 
  for each  $v \in (V(G) \setminus T) \cap \Gamma(v_{min})$ 
    do if  $W[v, v_{min}] < \beta[v]$  then  $\beta[v] \leftarrow W[v, v_{min}]; \alpha[v] \leftarrow v_{min}$ 
until  $T = V(G)$ 
return ( $\textit{Drzewo}$ )
```

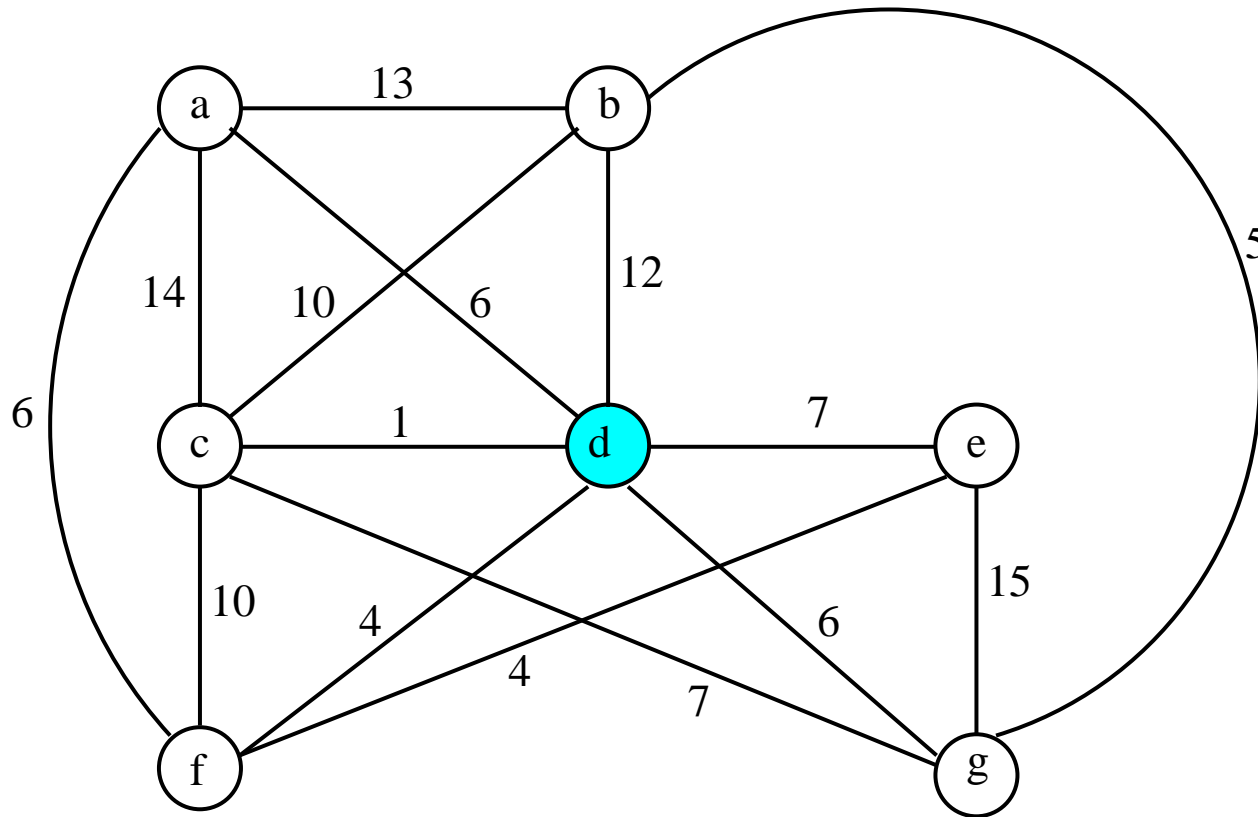
Złożoność obliczeniowa algorytmu Prima

- w powyższym pętla **repeat** wykonywana jest $n - 1$ razy, a każde wykonanie zakresu pętli ma złożoność $O(n)$; zatem opisany wyżej algorytm ma złożoność $O(n^2)$
- stosując bardziej wyrafinowane struktury danych (stertę) można jeszcze przyspieszyć ten algorytm do czasu $O(m \log n)$.

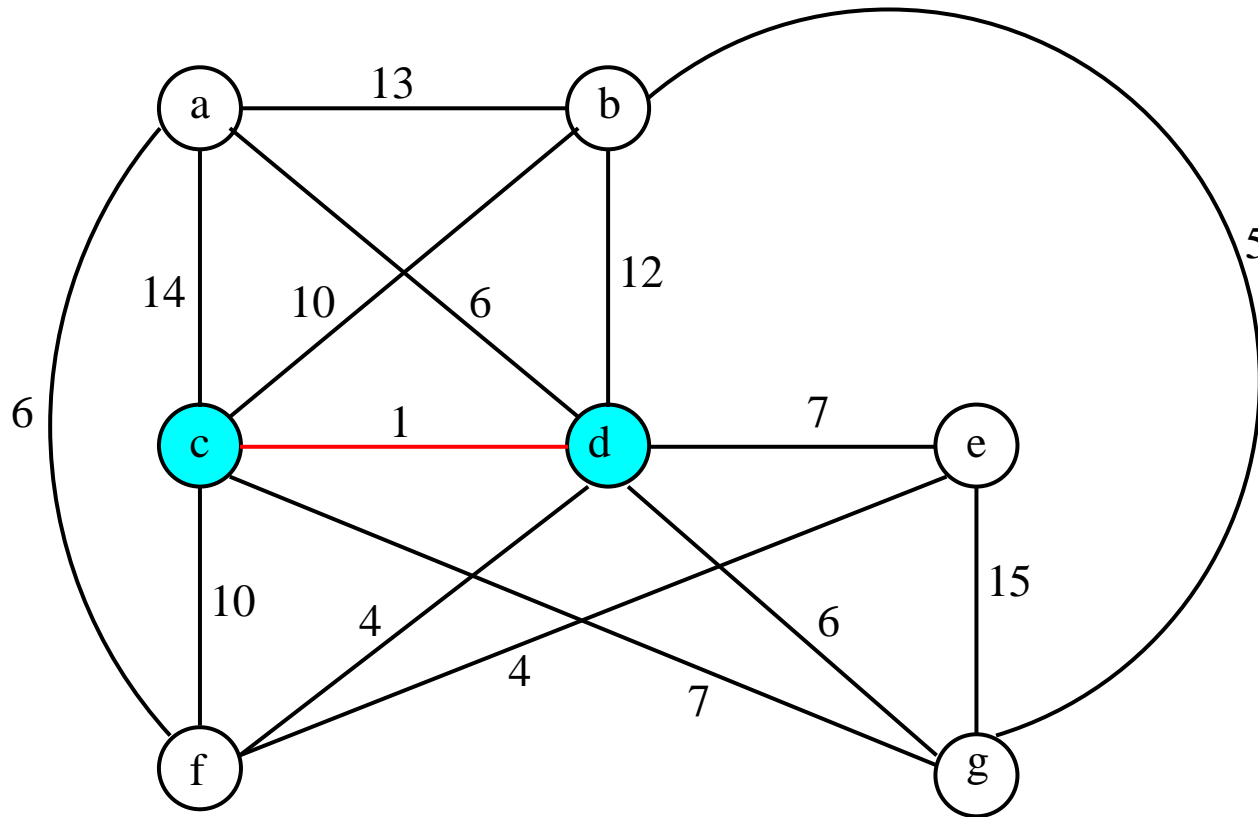
Przykład – algorytm Prima



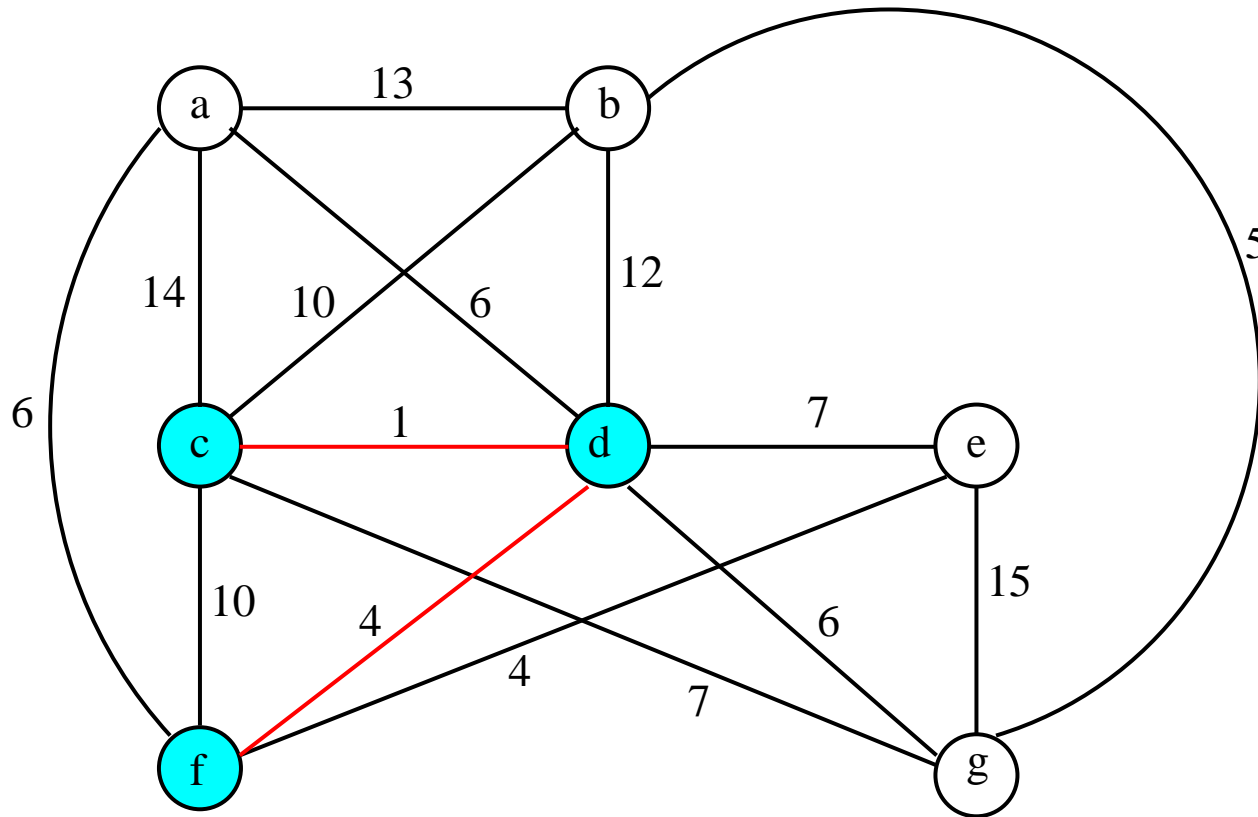
Przykład – algorytm Prima



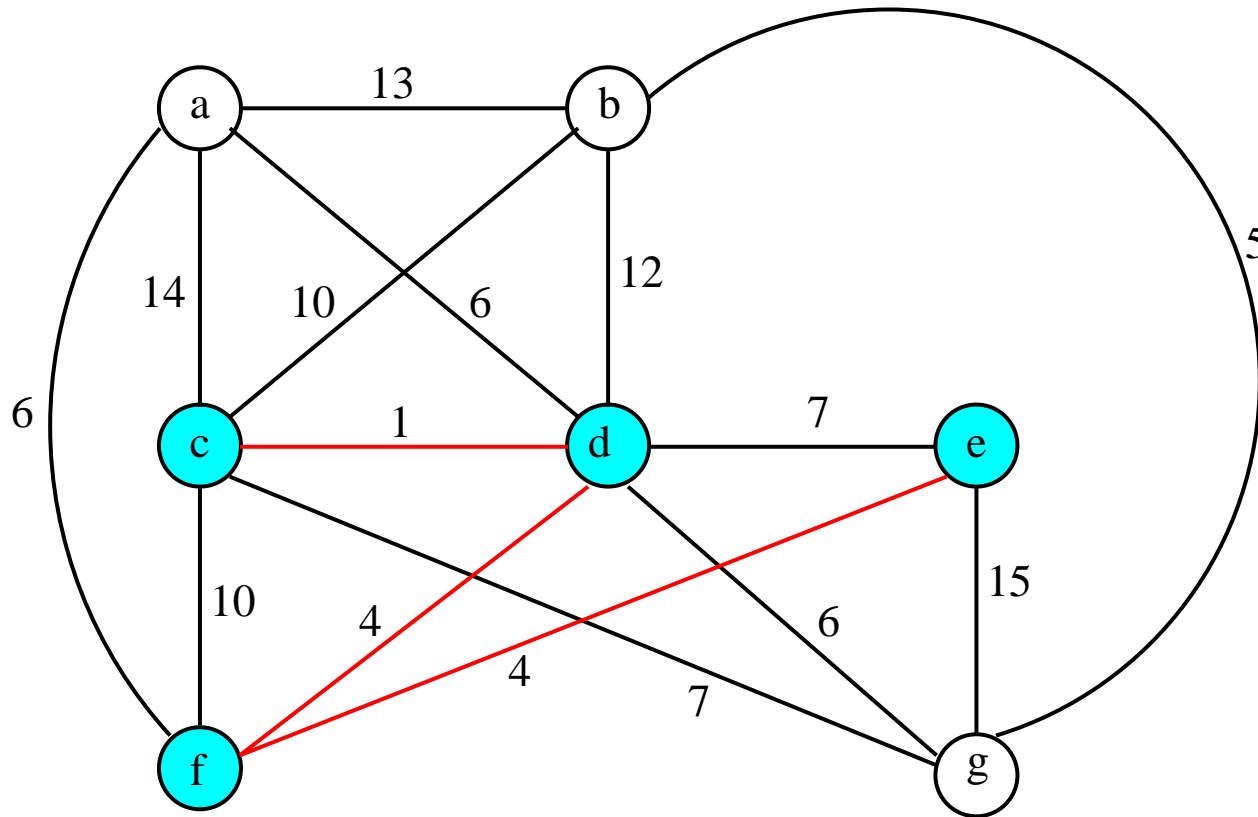
Przykład – algorytm Prima



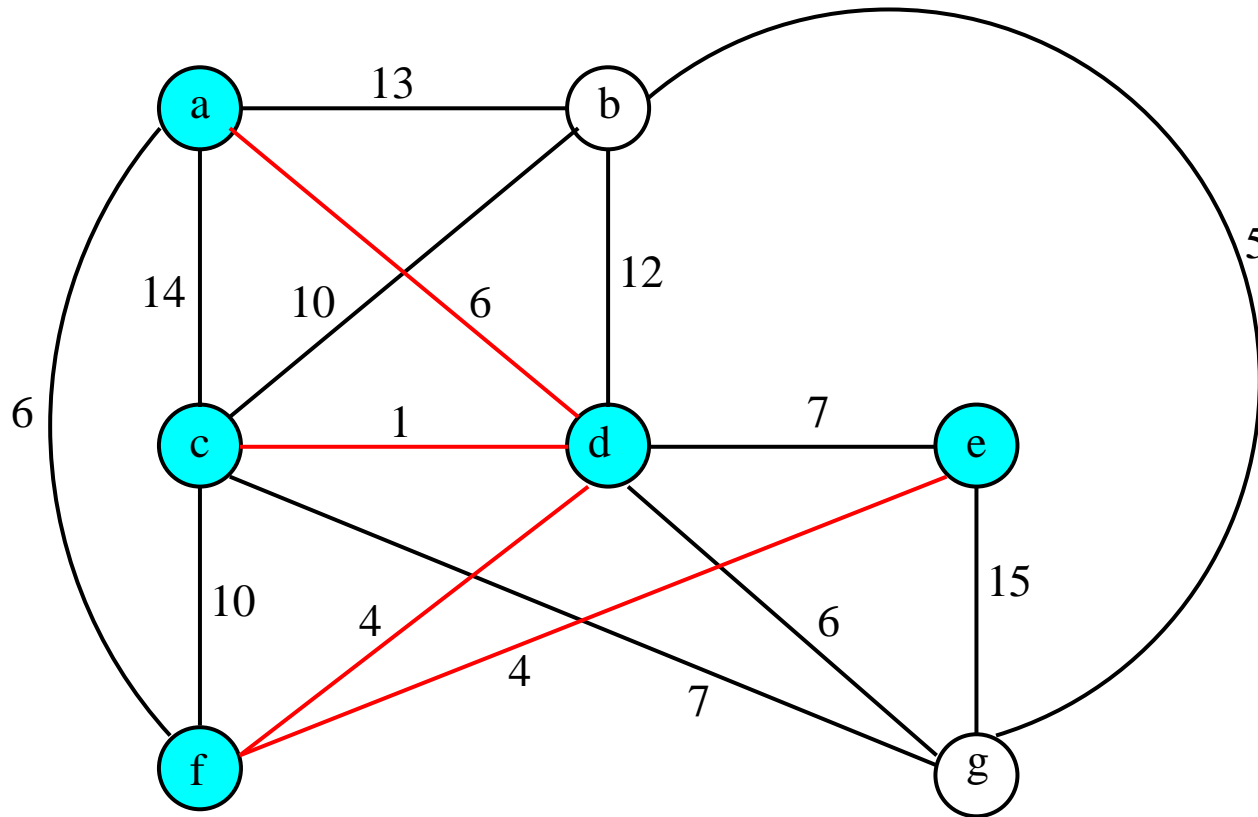
Przykład – algorytm Prima



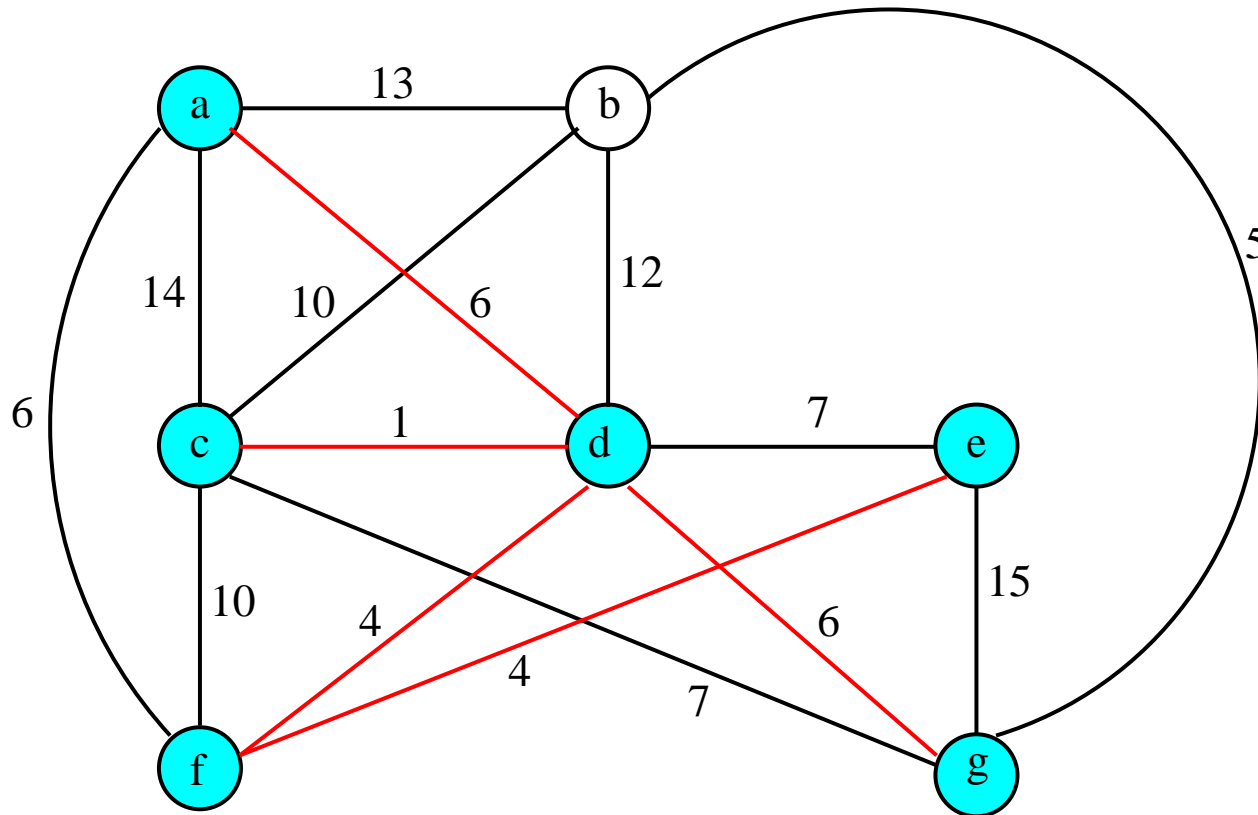
Przykład – algorytm Prima



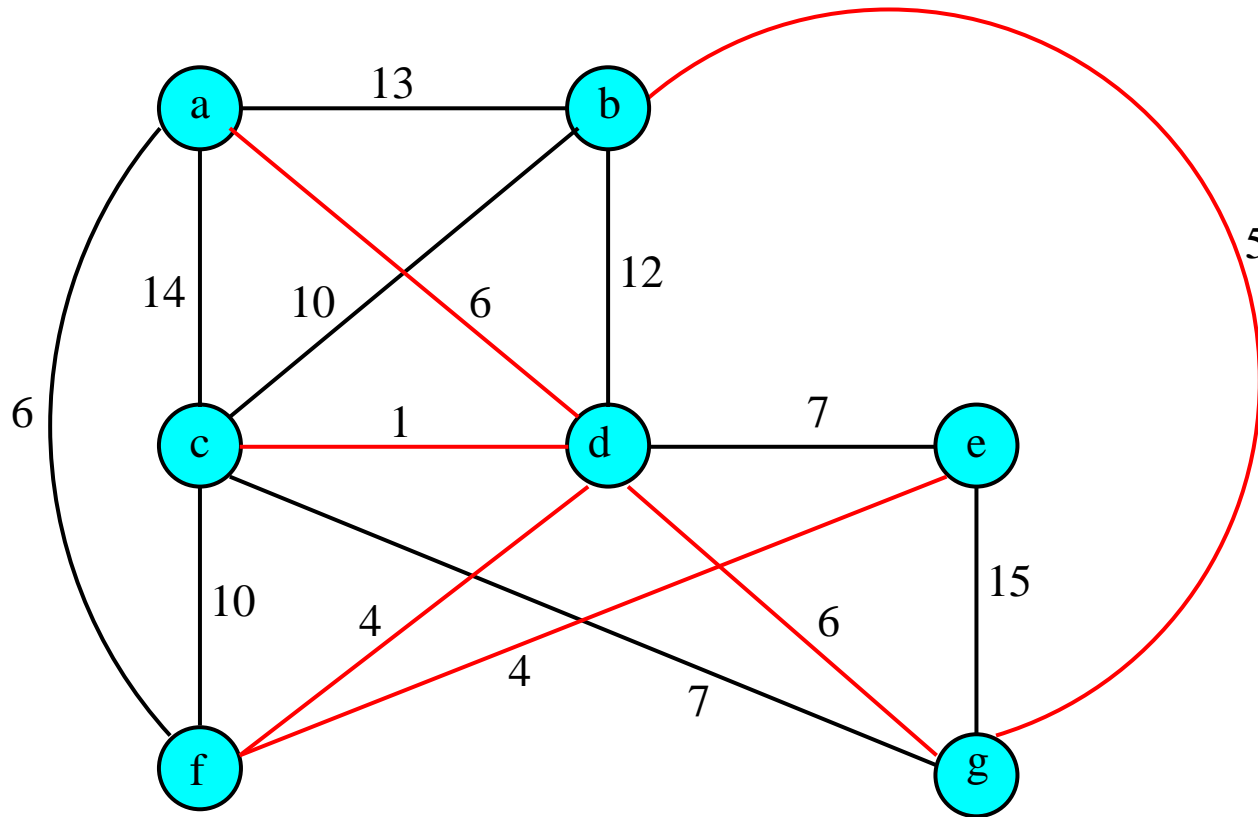
Przykład – algorytm Prima



Przykład – algorytm Prima



Przykład – algorytm Prima



Uwagi końcowe

- problem minimalnego drzewa rozpiętego to jeden z najstarszych problemów optymalizacji kombinatorycznej
- pierwszy algorytm MST - Borůvka (1926), Jarník (1930), Choquet (1938)
- polski wkład: Florek, Łukaszewicz, Perkal, Steinhaus, Zubrzycki (1951) - "taksonomia wrocławska"
- szereg ulepszeń standardowych algorytmów MST (o złożoności $O(m \log n)$): Yao (1975) i Cheriton, Tarjan (1976) ($O(m \log \log n)$), Fredman, Tarjan (1987) ($O(m \beta(m, n))$), gdzie $\beta(m, n)$ to najmniejsze $k \geq 1$ takie, że $\log^{(k)} m/n \leq 1$, Gabow, Galil, Spencer, Tarjan (1987) ($O(m \log \beta(m, n))$)

Twierdzenie (Chazelle, 2000). Minimalne rozpięte drzewo (MST) w grafie spójnym o n wierzchołkach i m krawędziach można wyznaczyć w czasie $O(m\alpha(m, n))$, gdzie $\alpha(m, n)$ jest odwrotnością funkcji Ackermana, to znaczy

$$\alpha(m, n) = \min\{k : \geq 1 : A(k, \lfloor m/n \rfloor) > \log n\}$$

, gdzie funkcja Ackermana jest zdefiniowana jako:

$A(1, j) = 2^j$, dla $j \geq 1$; $A(i, 1) = A(i - 1, 2)$, dla $i \geq 2$;
oraz $A(i, j) = A(i - 1, A(i, j - 1))$, dla $i, j \geq 2$.

- dodatkowe informacje o algorytmach MST oraz funkcji Ackermana – patrz książka Cormen, Leisserson i Rivest, odpowiednio Rozdział 24 (MST) oraz paragraf 22.4 (funkcja Ackermana)

7. Skojarzenia w grafach

7.1 Definicje i twierdzenia

Definicja (Skojarzenie w grafie). Podzbiór M zbioru krawędzi $E(G)$ nazywamy skojarzeniem grafu G jeżeli M nie zawiera pętli i żadne dwie krawędzie z M nie są przyległe. Dwa końce krawędzi z M są skojarzone przez M .

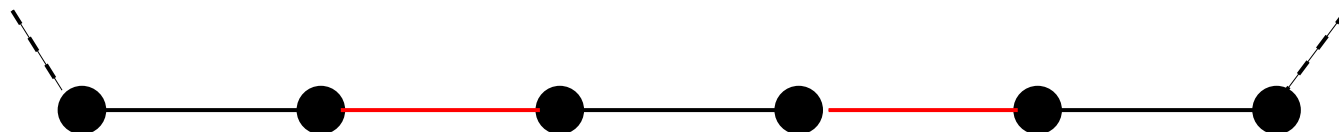
Mówimy, że skojarzenie M nasyca wierzchołek v (lub, że v jest M -nasycony) jeżeli pewna krawędź z M jest incydentna z v ; w przeciwnym razie wierzchołek v jest M -nienasycony.

Definicja (Skojarzenie doskonałe i największe). Jeżeli skojarzenie M nasyca każdy wierzchołek grafu G , to M nazywamy skojarzeniem doskonałym. Natomiast M jest skojarzeniem największym jeżeli graf G nie ma skojarzenia M' takiego, że $|M'| > |M|$.

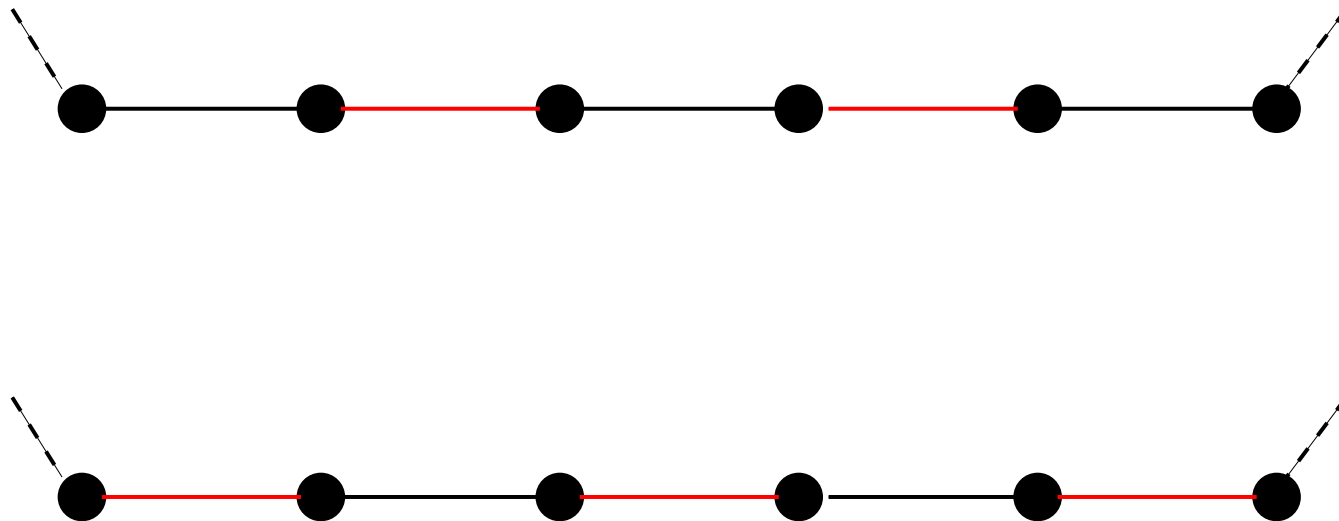
- Oczywiście każde skojarzenie doskonałe jest największe, chociaż nie zawsze skojarzenie największe jest doskonałe

Definicja . Ścieżkę, do której należą na przemian krawędzie z M i M^c , gdzie $M^c = E(G) - M$, nazywać będziemy ścieżką M -przemianną. Jeżeli początek i koniec M -przemiennej ścieżki jest M -nienasycony, to nazywamy ją ścieżką M -powiększającą.

Twierdzenie (Petersen, Berge). Skojarzenie M w grafie G jest największe wtedy i tylko wtedy, gdy G nie zawiera ścieżki M -powiększającej.



Twierdzenie (Petersen, Berge). Skojarzenie M w grafie G jest największe wtedy i tylko wtedy, gdy G nie zawiera ścieżki M -powiększającej.



7.2 Skojarzenia w grafach dwudzielnych

- klasyczny mini-maxowy związek podany przez Königa (1931) charakteryzuje rozmiar (moc) **największego** skojarzenia ($\alpha'(G)$) w grafie dwudzielnym

Definicja . Podzbiór U zbioru $V(G)$ nazywamy pokryciem wierzchołkowym (krawędzi) jeżeli każda krawędź grafu G ma przynajmniej jeden koniec w U (moc **najmniejszego** pokrycia wierzchołkowego oznaczamy przez $\beta(G)$).

Twierdzenie (König, 1931). W grafie dwudzielnym G moc **największego** skojarzenia jest równa mocy **najmniejszego** pokrycia wierzchołkowego, tzn.

$$\alpha'(G) = \beta(G)$$

Wniosek (Twierdzenie Frobeniusa, 1917). *Dwudzielny graf G ma skojarzenie doskonałe wtedy i tylko wtedy gdy każde pokrycie wierzchołkowe ma rozmiar co najmniej $|V(G)|/2$.*

Dowód. Wynika bezpośrednio z twierdzenia Königa, ponieważ graf G ma skojarzenie doskonałe wtedy i tylko wtedy gdy $\alpha'(G) \geq |V|/2$. ■

Wniosek (Twierdzenie o parach małżeńskich). *Jeżeli G jest k -regularnym grafem dwudzielnym ($k > 0$), to G ma skojarzenie doskonałe.*

Dowód. Jeżeli graf G jest k -regularnym grafem dwudzielnym to każdy jego wierzchołek jest incydentny z dokładnie k krawędziami. Ponieważ $|E(G)| = k|V(G)|/2$, więc potrzebujemy co najmniej $|V(G)|/2$ wierzchołków aby pokryć wszystkie krawędzie. Zatem z twierdzenia Frobeniusa wynika, że w G istnieje skojarzenie doskonałe. ■

Algorytm węgierski

Dane: graf dwudzielnny G ($V = (X, Y)$) oraz skojarzenie M

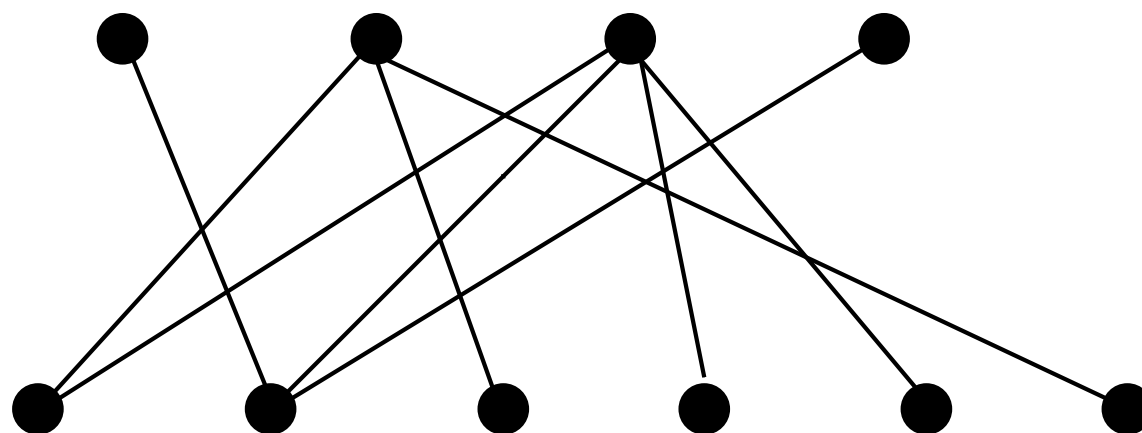
Poszukiwane: największe skojarzenie w grafie G

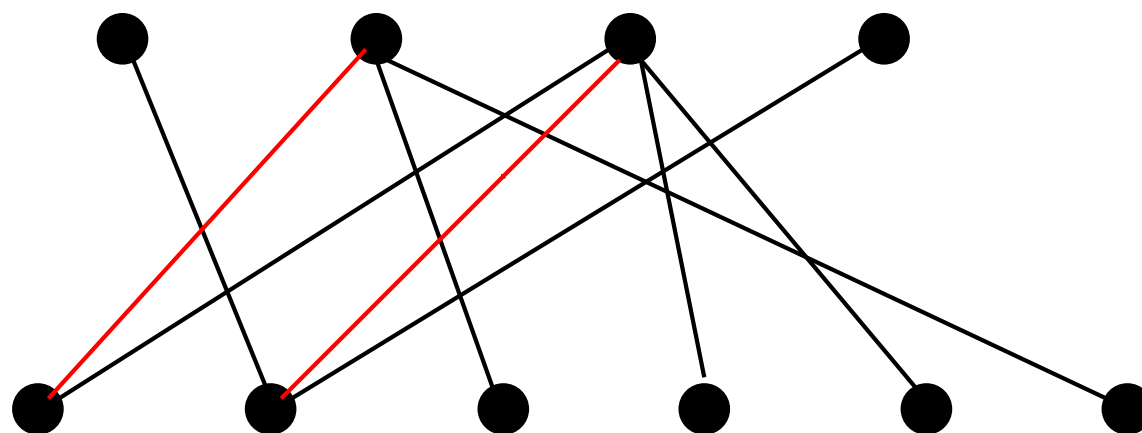
1. Zorientuj każdą krawędź $e = xy$, gdzie $x \in X$, $y \in Y$ grafu G w następujący sposób:
 - (a) jeżeli $e \in M$ to zorientuj e od y do x
 - (b) jeżeli $e \notin M$ to zorientuj e od x do y
2. Znajdź zbiory X_M i Y_M wierzchołków M -nienasyconych w zbiorach, odpowiednio, X i Y .

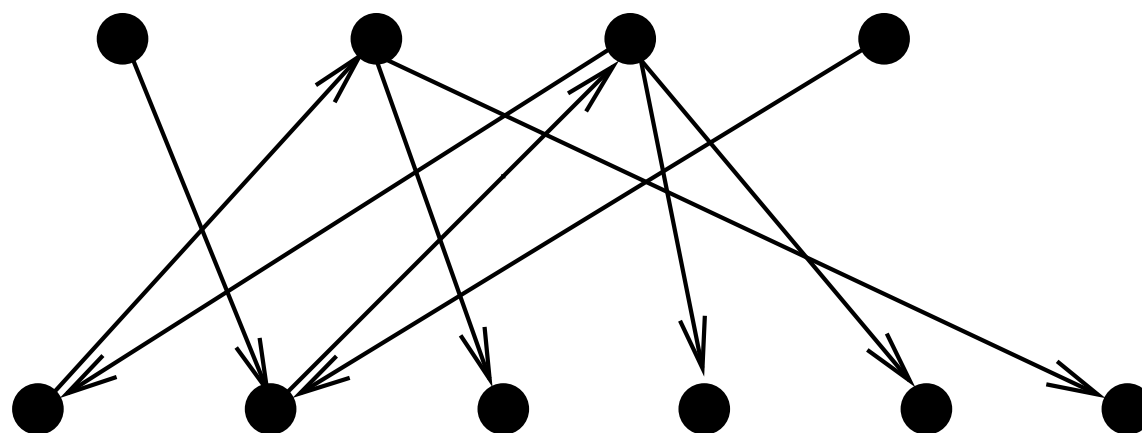
3 Znajdź ścieżkę M -powiększającą P , poprzez wyznaczenie skierowanej ścieżki (o ile taka istnieje) z X_M do Y_M , $M \leftarrow M \div E(P)$ i przejdź do kroku 1, w przeciwnym razie STOP.

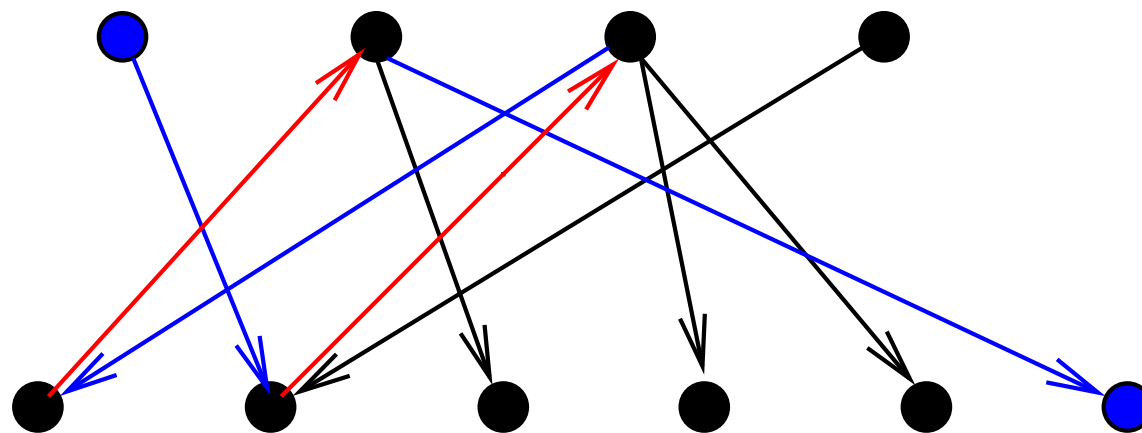
Twierdzenie . *Największe skojarzenie w grafie dwudzielnym można znaleźć w czasie $O(nm)$.*

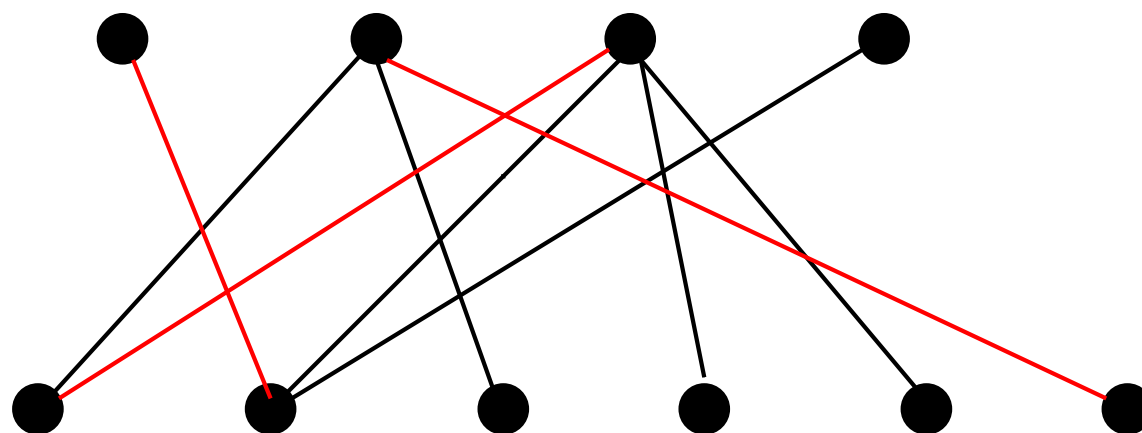
Dowód. Zauważmy, że w powyższym algorytmie mamy co najwyżej n iteracji, z których każda może być wykonana, za pomocą algorytmu BFS, w czasie $O(m)$ ■











Algorithm 0.0.1: METODA WĘGIERSKA($G = ((X, Y), E)$)

$M \leftarrow \emptyset$

repeat

$X_M \leftarrow Y_M \leftarrow \emptyset$

for each $e = xy \in E$

do $\begin{cases} X_M \leftarrow X_M \cup \{x\}; & Y_M \leftarrow Y_M \cup \{y\} \\ \text{if } e \in M \\ \quad \text{then } e \leftarrow \overrightarrow{yx} \\ \quad \text{else } e \leftarrow \overrightarrow{xy} \end{cases}$

$X_M \leftarrow X \setminus X_M; \quad Y_M \leftarrow Y \setminus Y_M$

$exists \leftarrow$ czy istnieje skierowana ścieżka z X_M do Y_M

if $exists$

then $\begin{cases} P \leftarrow \text{dowolna skierowana ścieżka z } X_M \text{ do } Y_M \\ M \leftarrow M \div E(P) \end{cases}$

until not $exists$

return (M)

- Kolejny algorytm dowodzi, że można poprawić czas znajdowania skojarzenia o największej mocy do następujących wartości:

Twierdzenie . *Największe skojarzenie w grafie dwudzielnym można znaleźć w czasie $O(n^{1/2}m)$.*

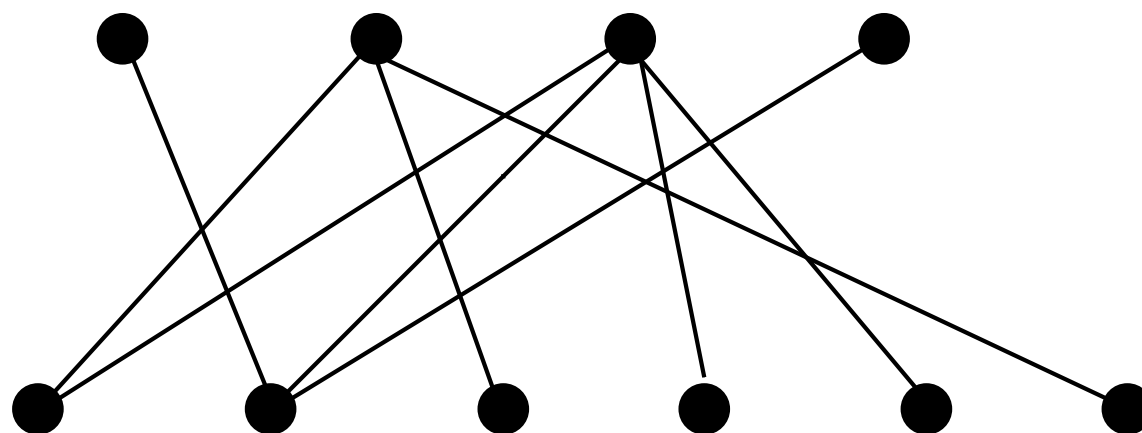
Twierdzenie . *Największe skojarzenie w grafie dwudzielnym można znaleźć w czasie $O(\beta(G)^{1/2}m) = O(\alpha'(G)^{1/2}m)$.*

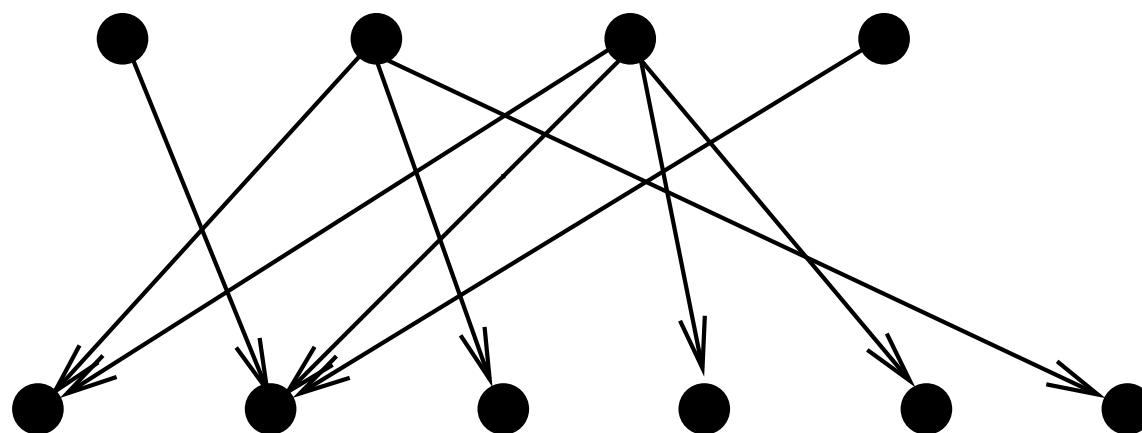
Algorytm “ścieżkowy”

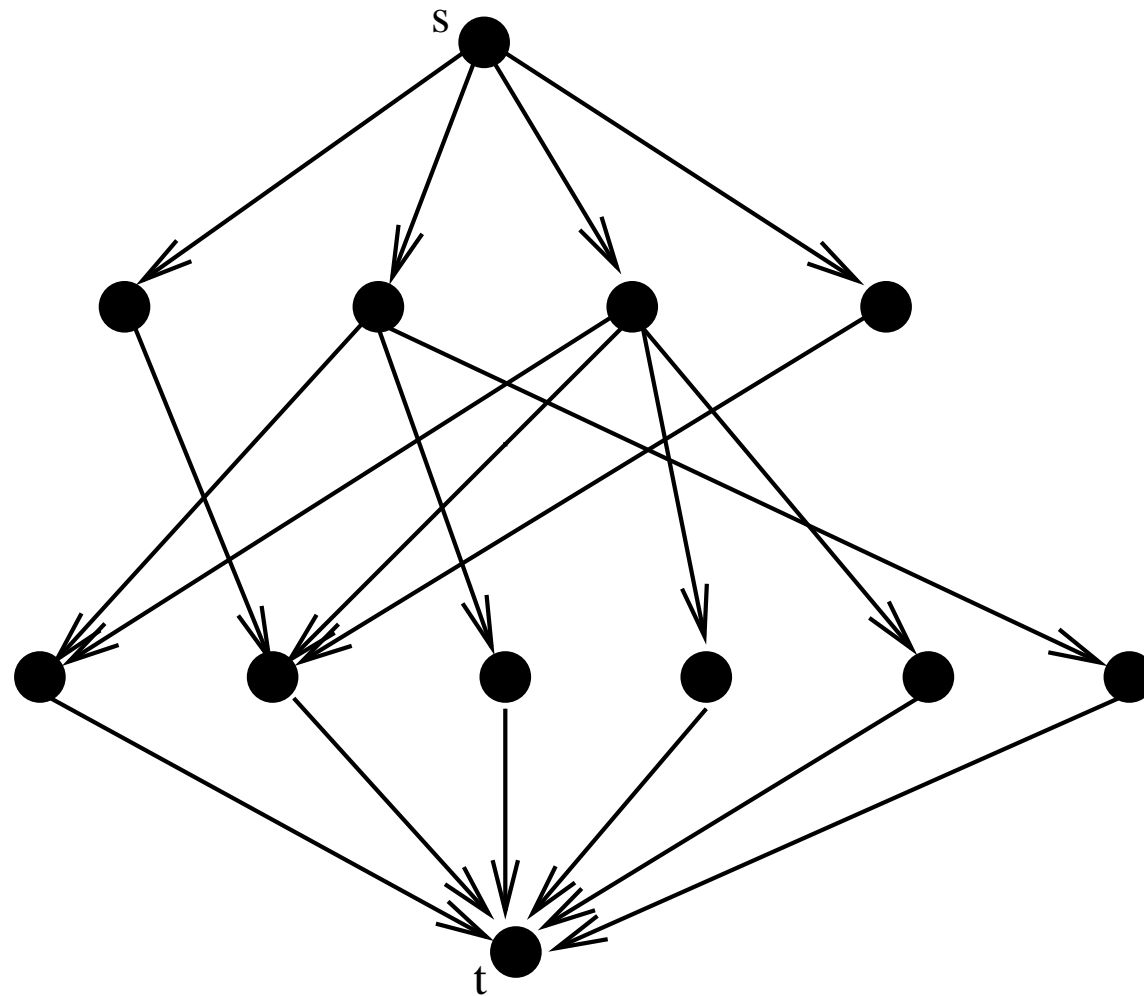
Dane: graf dwudzielny G ($V = (X, Y)$)

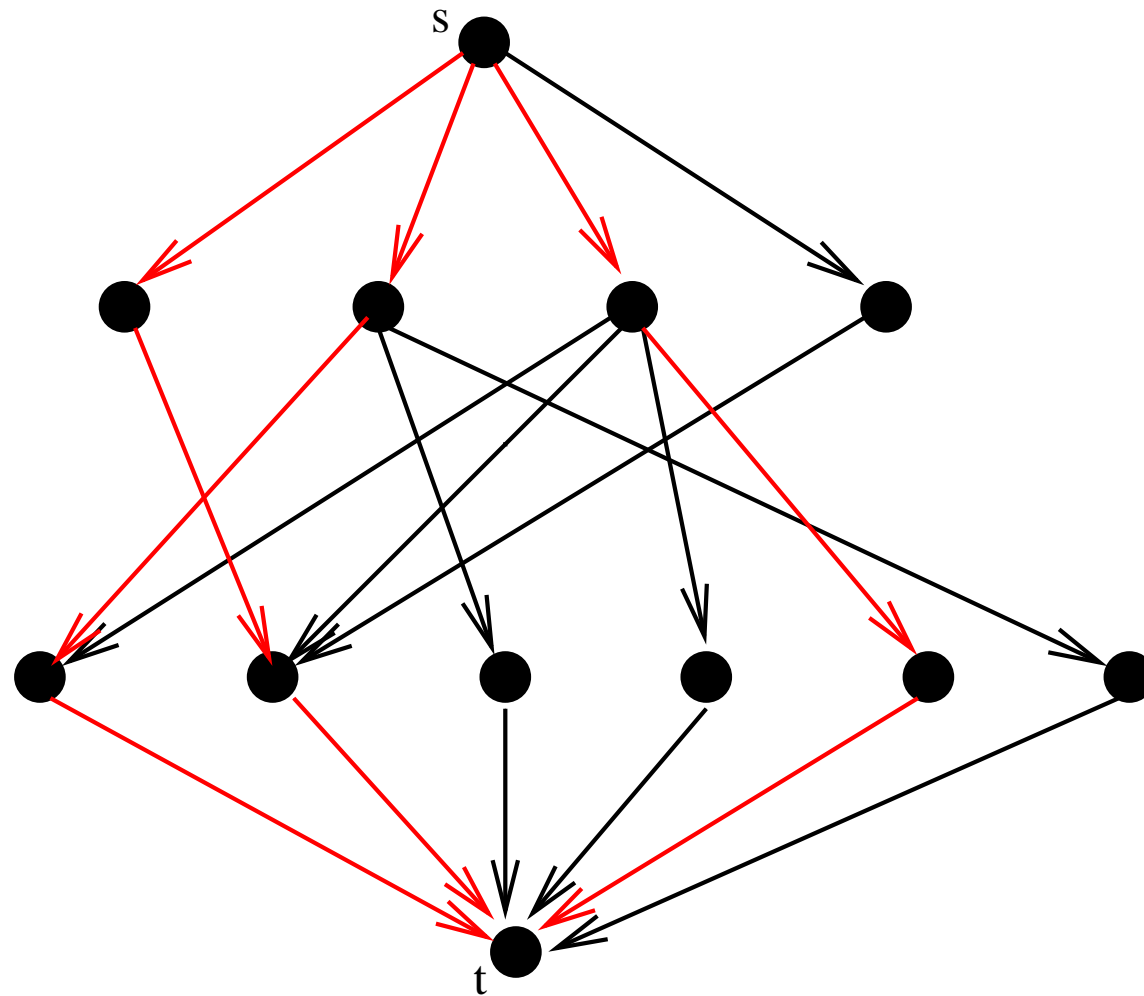
Poszukiwane: skojarzenie M o największej mocy w G

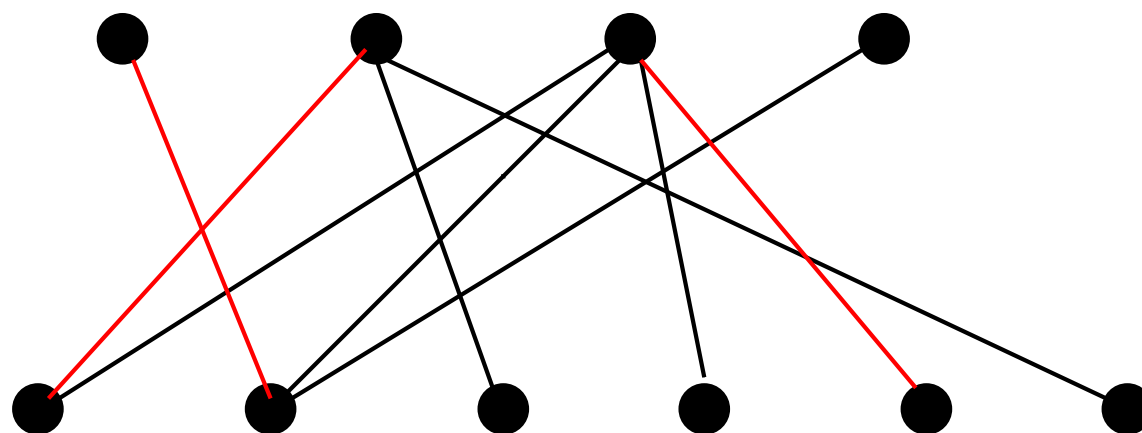
1. Utwórz z grafu G digraf D poprzez nadanie wszystkim krawędziom grafu G orientacji od zbioru X do zbioru Y .
2. Dodaj dwa nowe wierzchołki s oraz t i nowe łuki: od wierzchołka s do wszystkich wierzchołków z X , oraz od każdego wierzchołka z Y do wierzchołka t .
3. Znajdź wszystkie wewnętrznie wierzchołkowo rozłączne ścieżki z s do t (w czasie $O(n^{1/2}m)$ lub $O(\beta(G)^{1/2}m)$)
4. Zalicz do M wszystkie krawędzie grafu G należące do znalezionej rodziny ścieżek.











Algorithm 0.0.1: METODAŚCIEŻKOWA($G = ((X, Y), E)$)

$V_D \leftarrow X \cup Y \cup \{s, t\}$

$E_D \leftarrow E$

for each $x \in X$

do $E_D \leftarrow E_D \cup \{sx\}$

for each $y \in Y$

do $E_D \leftarrow E_D \cup \{ys\}$

$S \leftarrow$ wszystkie wewnętrznie wierzchołkowo rozłączne (s, t) -ścieżki w D

$M \leftarrow S \cap E$

return (M)

Dla każdego zbioru $S \subset V(G)$ przez **zbiór jego sąsiadów** $N_G(S)$ rozumiemy zbiór wszystkich wierzchołków przyległych do wierzchołków z S .

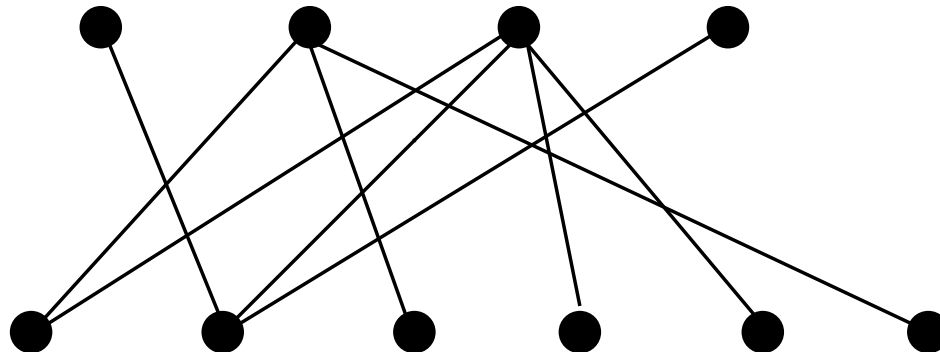
Twierdzenie (Hall). *Niech G będzie grafem dwudzielnym z dwupodziałem (X, Y) . Graf G zawiera skojarzenie, które nasycza każdy wierzchołek w X wtedy i tylko wtedy, gdy*

$$|N_G(S)| \geq |S| \quad \text{dla każdego} \quad S \subset X$$

Dla każdego zbioru $S \subset V(G)$ przez **zbiór jego sąsiadów** $N_G(S)$ rozumiemy zbiór wszystkich wierzchołków przyległych do wierzchołków z S .

Twierdzenie (Hall). Niech G będzie grafem dwudzielnym z dwupodziałem (X, Y) . Graf G zawiera skojarzenie, które nasycza każdy wierzchołek w X wtedy i tylko wtedy, gdy

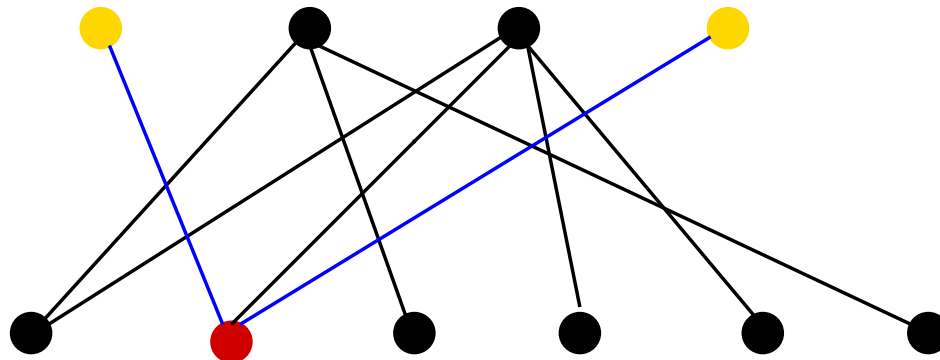
$$|N_G(S)| \geq |S| \quad \text{dla każdego} \quad S \subset X$$



Dla każdego zbioru $S \subset V(G)$ przez **zbiór jego sąsiadów** $N_G(S)$ rozumiemy zbiór wszystkich wierzchołków przyległych do wierzchołków z S .

Twierdzenie (Hall). Niech G będzie grafem dwudzielnym z dwupodziałem (X, Y) . Graf G zawiera skojarzenie, które nasycza każdy wierzchołek w X wtedy i tylko wtedy, gdy

$$|N_G(S)| \geq |S| \quad \text{dla każdego} \quad S \subset X$$



Problem przydziału zadań

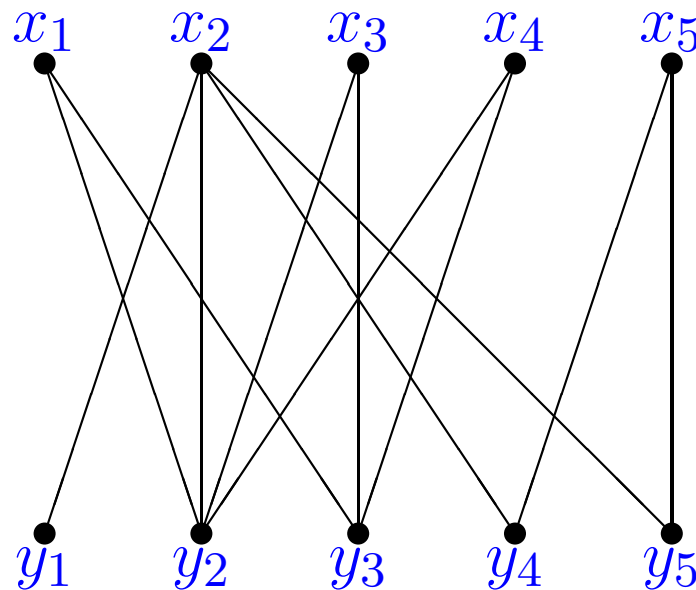
- Problem: znajdź w grafie dwudzielnym z $V = (X, Y)$ skojarzenie nasycające każdy wierzchołek X lub, jeżeli nie jest to możliwe, znajdź podzbiór S zbioru X taki, że $|N_G(S)| < |S|$.

Dane: graf dwudzielnym G ($V = (X, Y)$) oraz skojarzenie M

1. Jeżeli M nasycza wszystkie wierzchołki X , to STOP. W przeciwnym razie niech u , ($u \in X$) będzie M -nienasycony. $S \leftarrow \{u\}$, $T \leftarrow \emptyset$.
2. Jeżeli $N_G(S) = T$, to $|N_G(S)| < |S|$ i STOP - nie ma szukanego skojarzenia. W przeciwnym razie niech $y \in N_G(S) - T$.

3 Jeżeli y jest M -nasycony, $yz \in M$, to $S \leftarrow S \cup \{z\}$,
 $T \leftarrow T \cup \{y\}$ i przejdź do kroku 2. W przeciwnym razie
niech P będzie M -przemienną ścieżką powiększającą z
 u do y , $M \leftarrow M \div E(P)$ i przejdź do kroku 1.

Przykład . Wyznacz skojarzenie doskonałe, jeżeli takie istnieje, w grafie:



Algorithm 0.0.1: PRZYDZIAŁ ZADAŃ($G = ((X, Y), E)$)

$M \leftarrow \emptyset$

while $|M| \neq |X|$

$u \leftarrow M$ -nienasycony wierzchołek z X

$S \leftarrow \{u\}; T \leftarrow \emptyset$

repeat

if $N_G(S) = T$

then return (“nie ma szukanego skojarzenia”)

do $y \leftarrow$ dowolny element z $N_G(S) - T$

if $y \in M$ -nasycone

then $S \leftarrow S \cup \{z\}; T \leftarrow T \cup \{y\}$

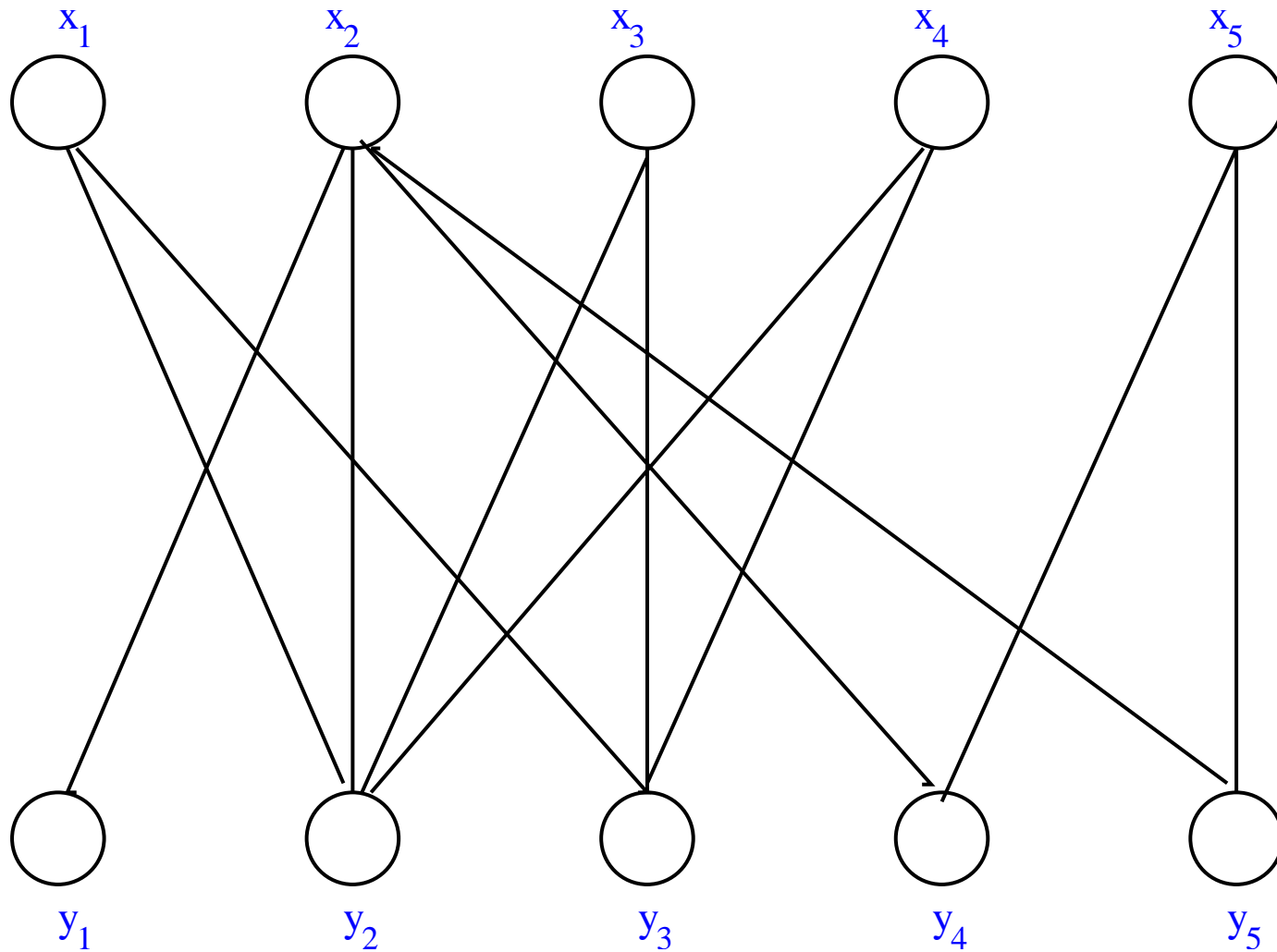
until $y \in M$ -nienasycone

$P \leftarrow M$ -przemienna ścieżka powiększającą z u do y

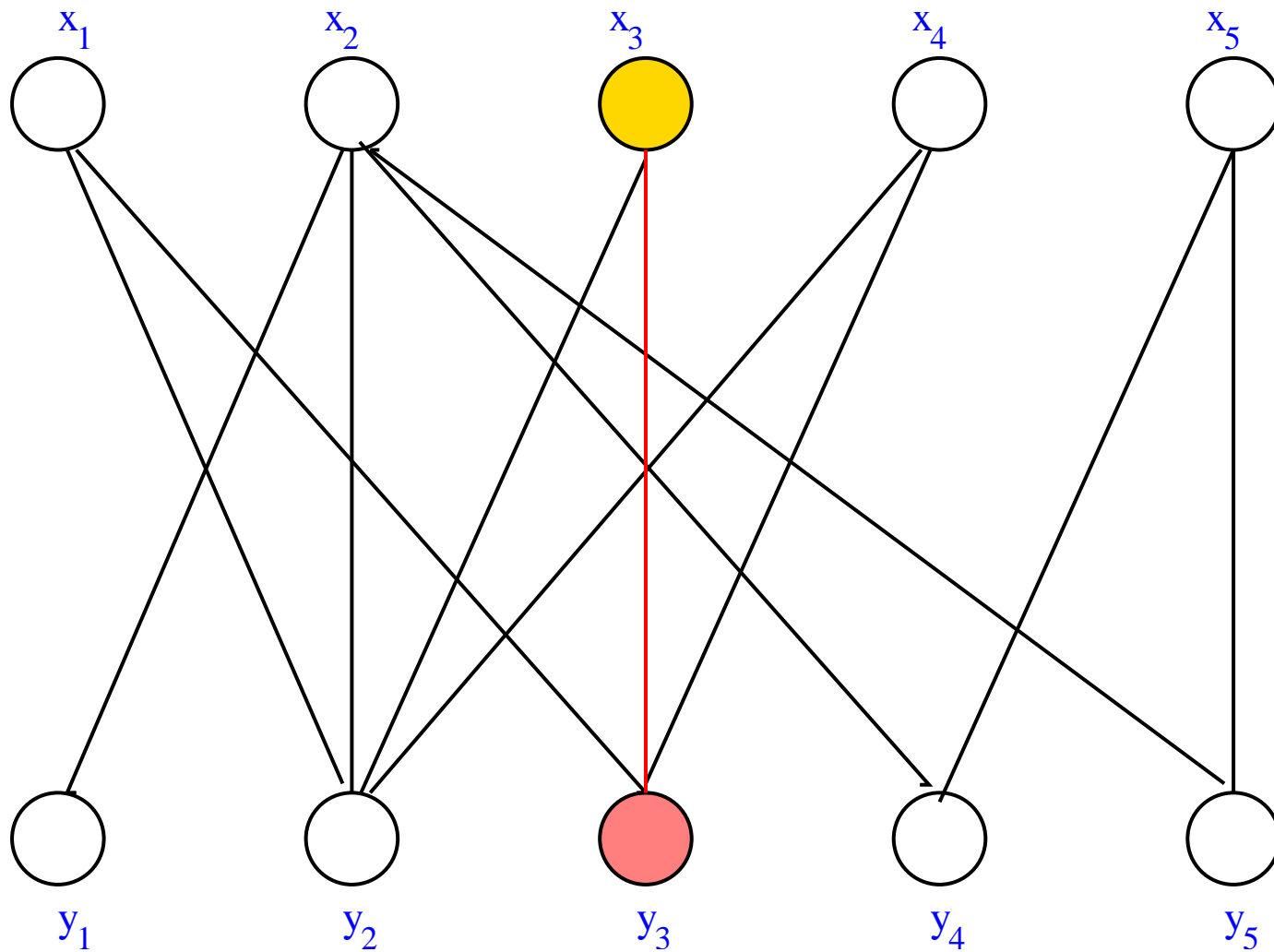
$M \leftarrow M \div E(P)$

return (M)

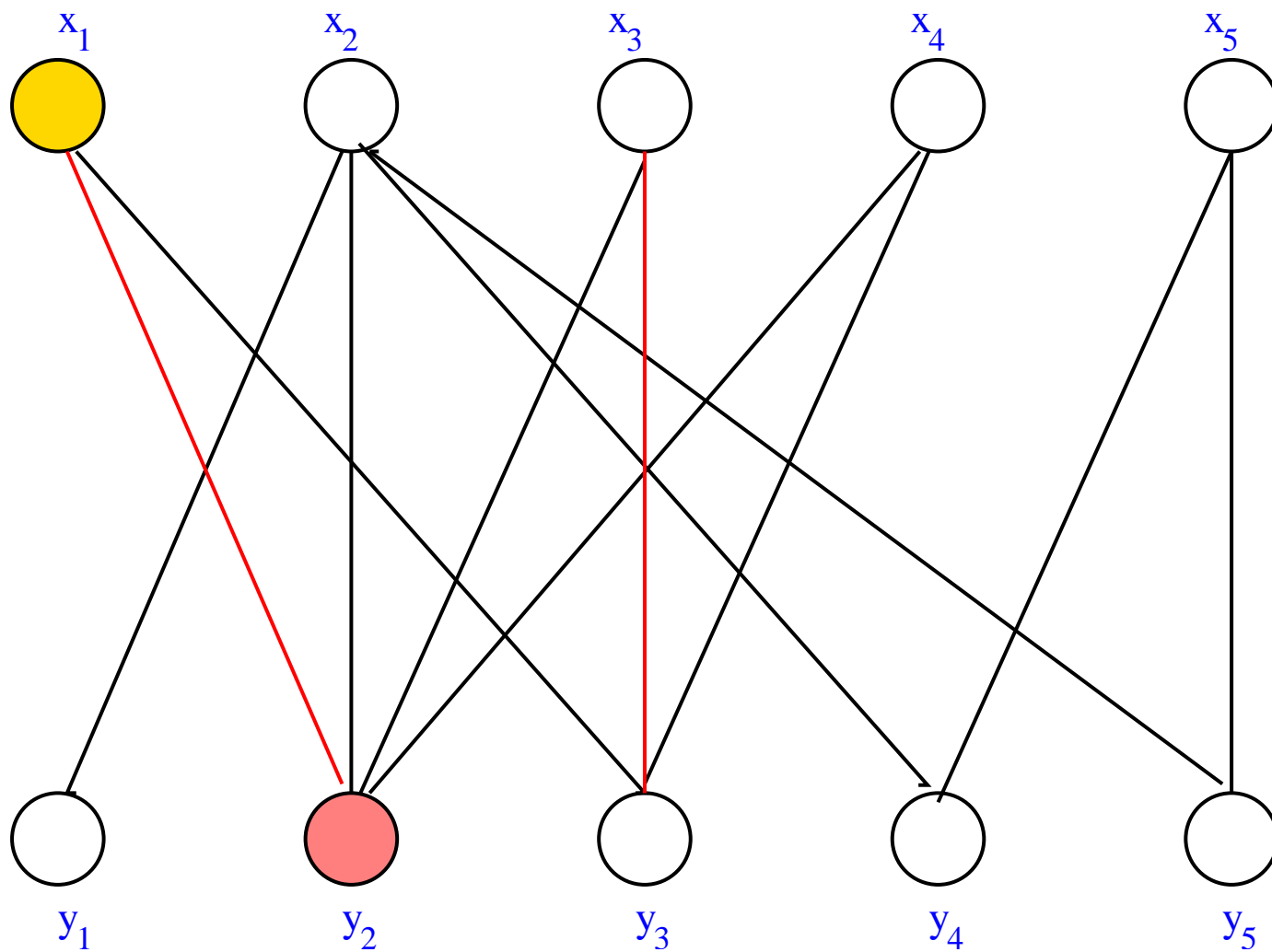
Przykład



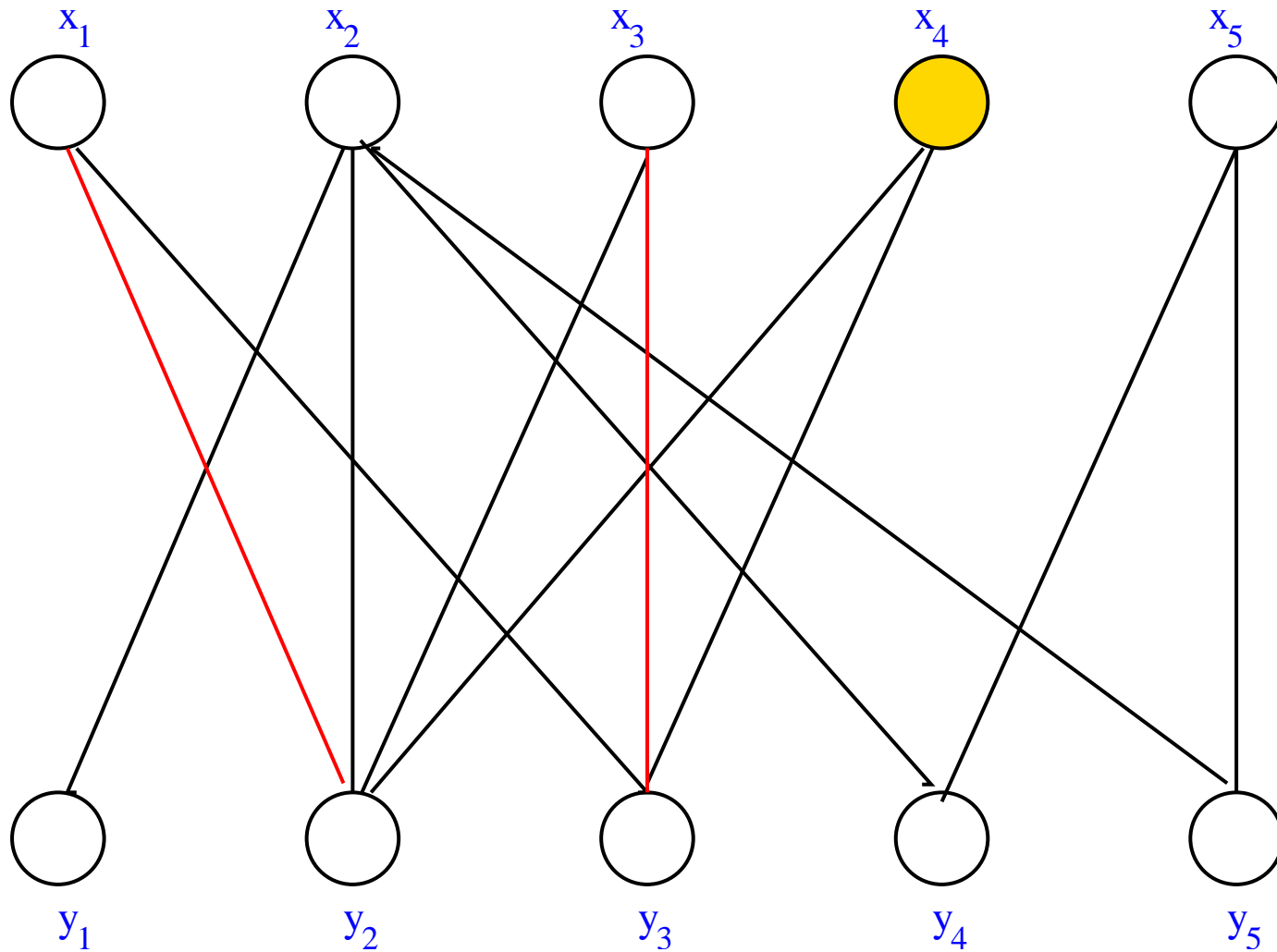
Przykład



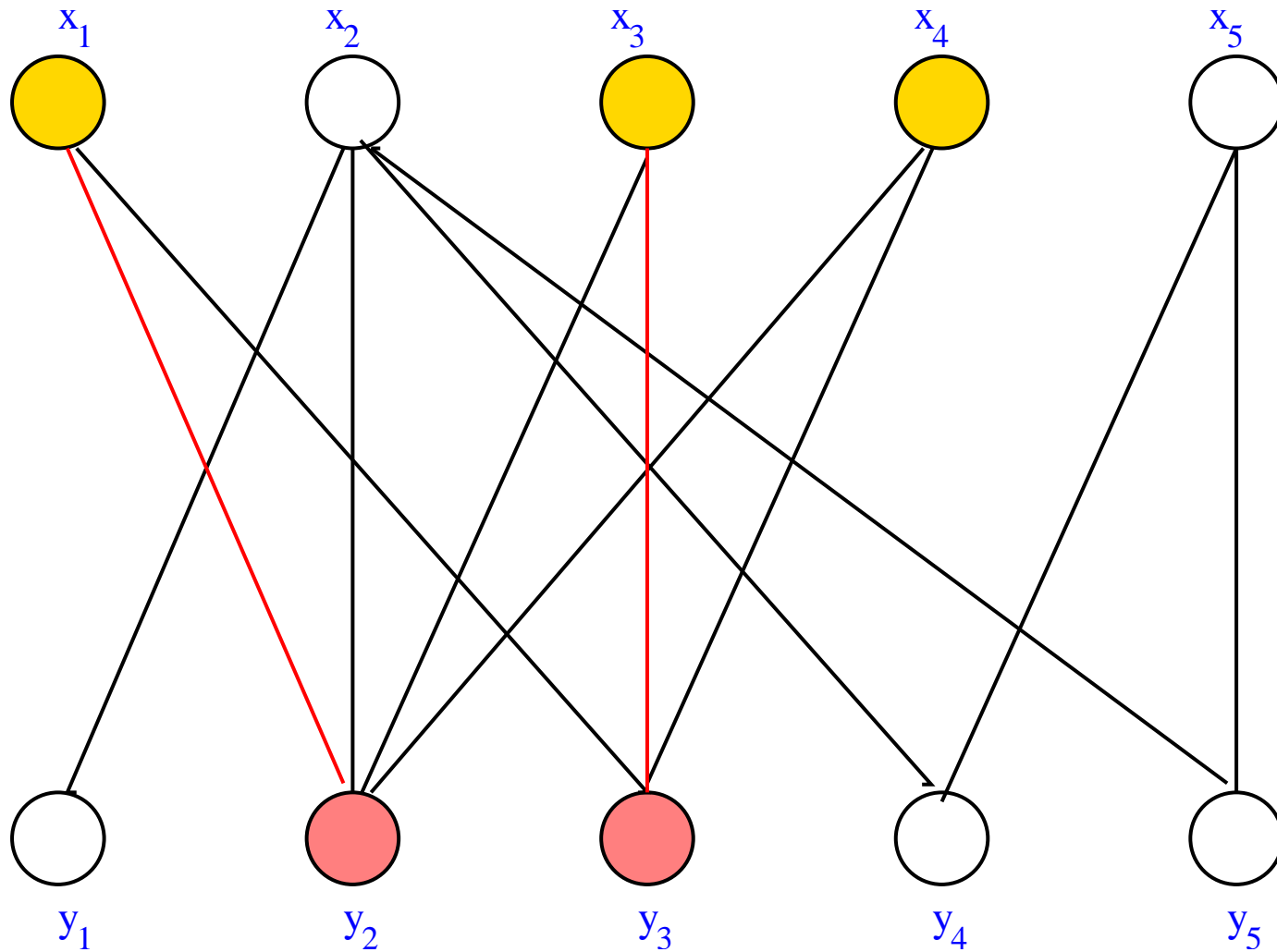
Przykład



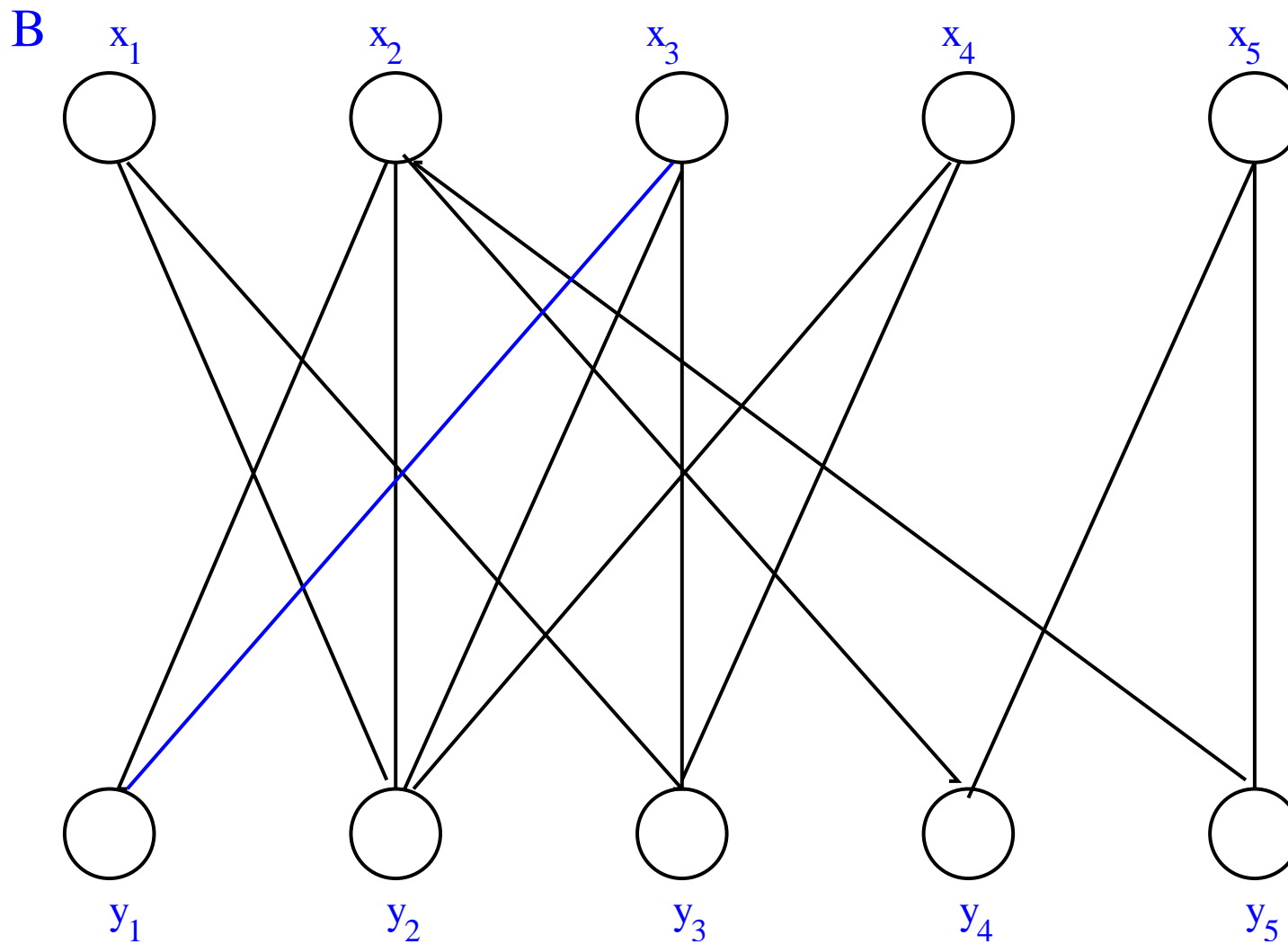
Przykład



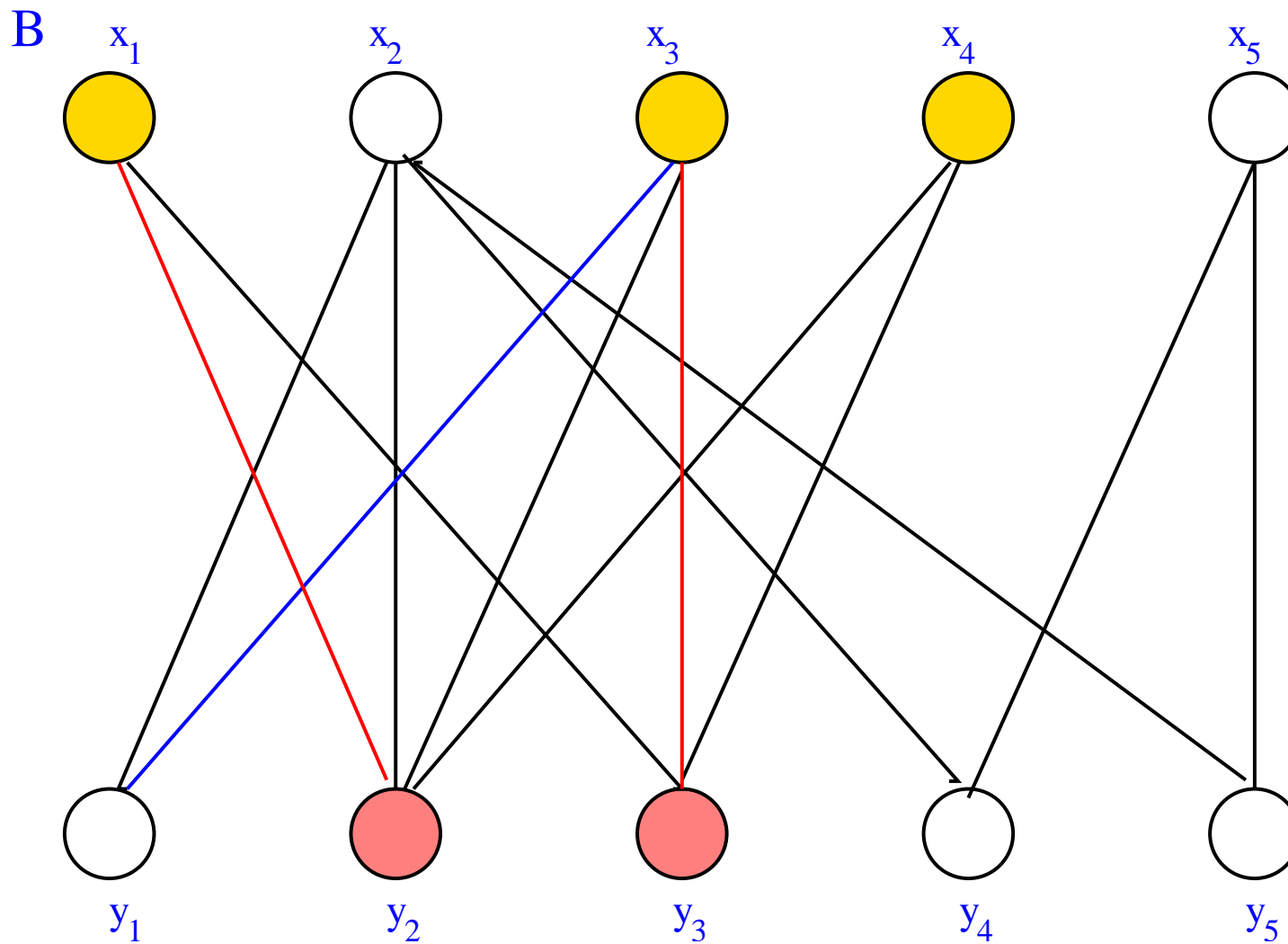
Przykład



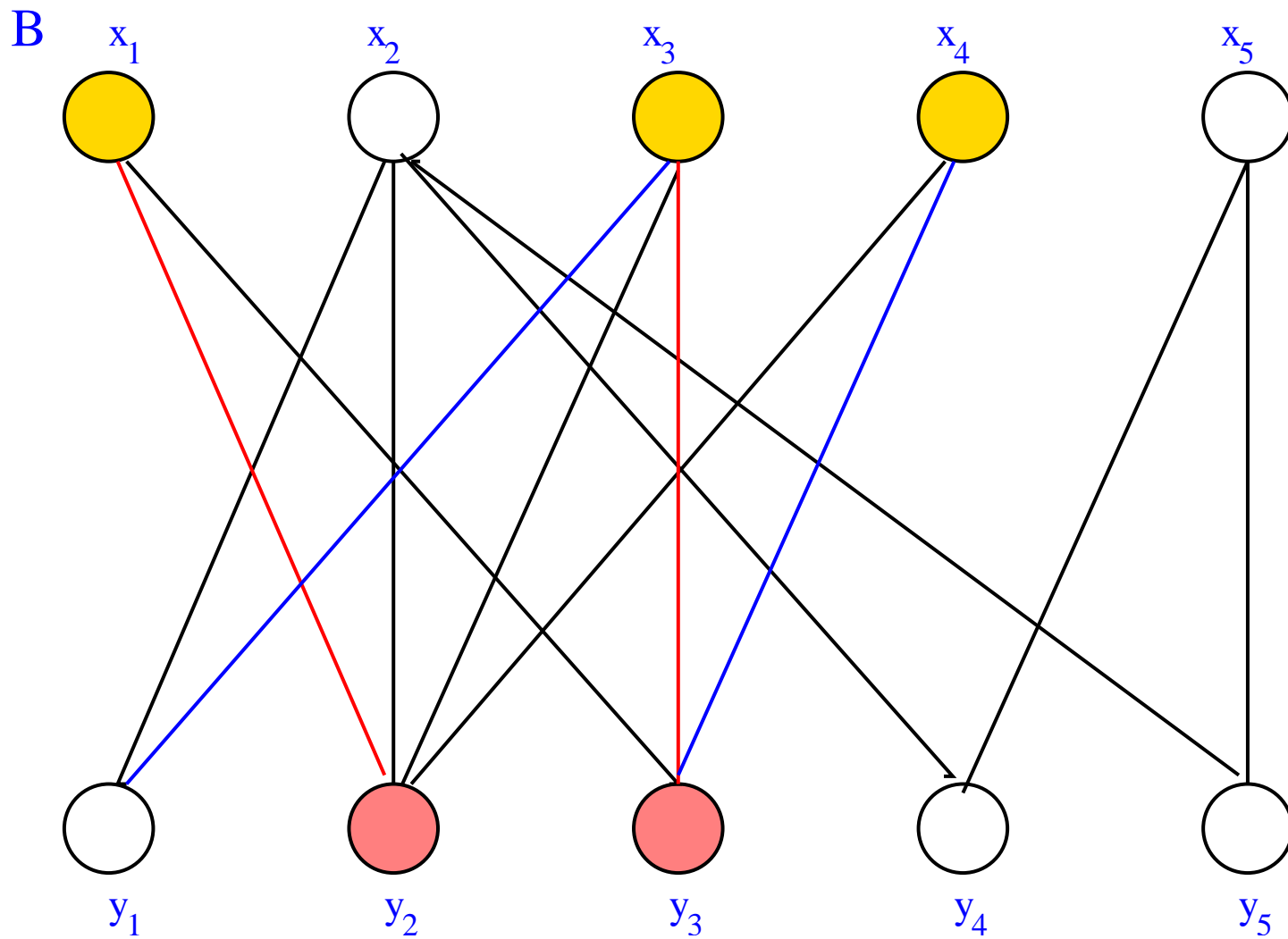
Przykład



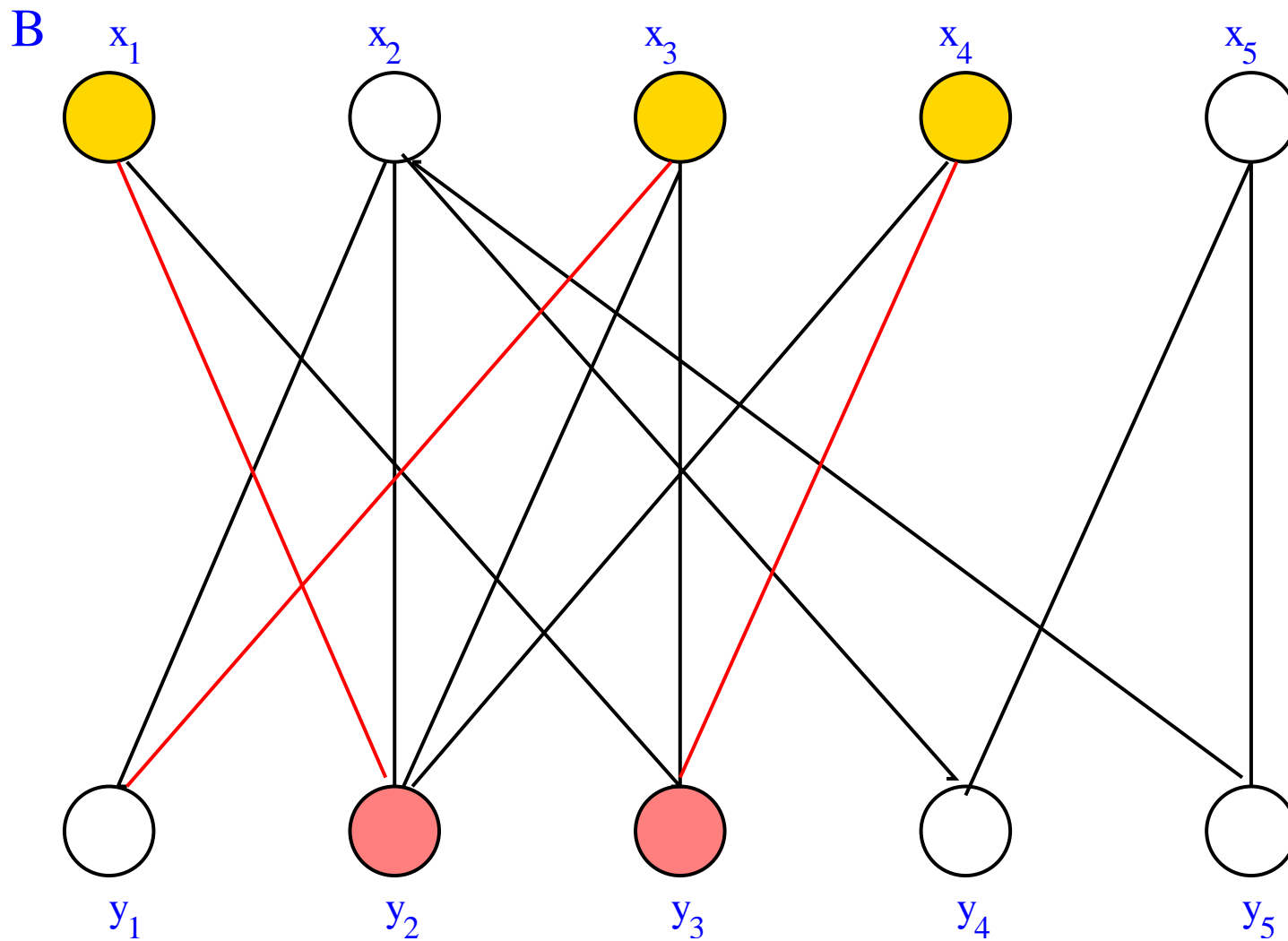
Przykład



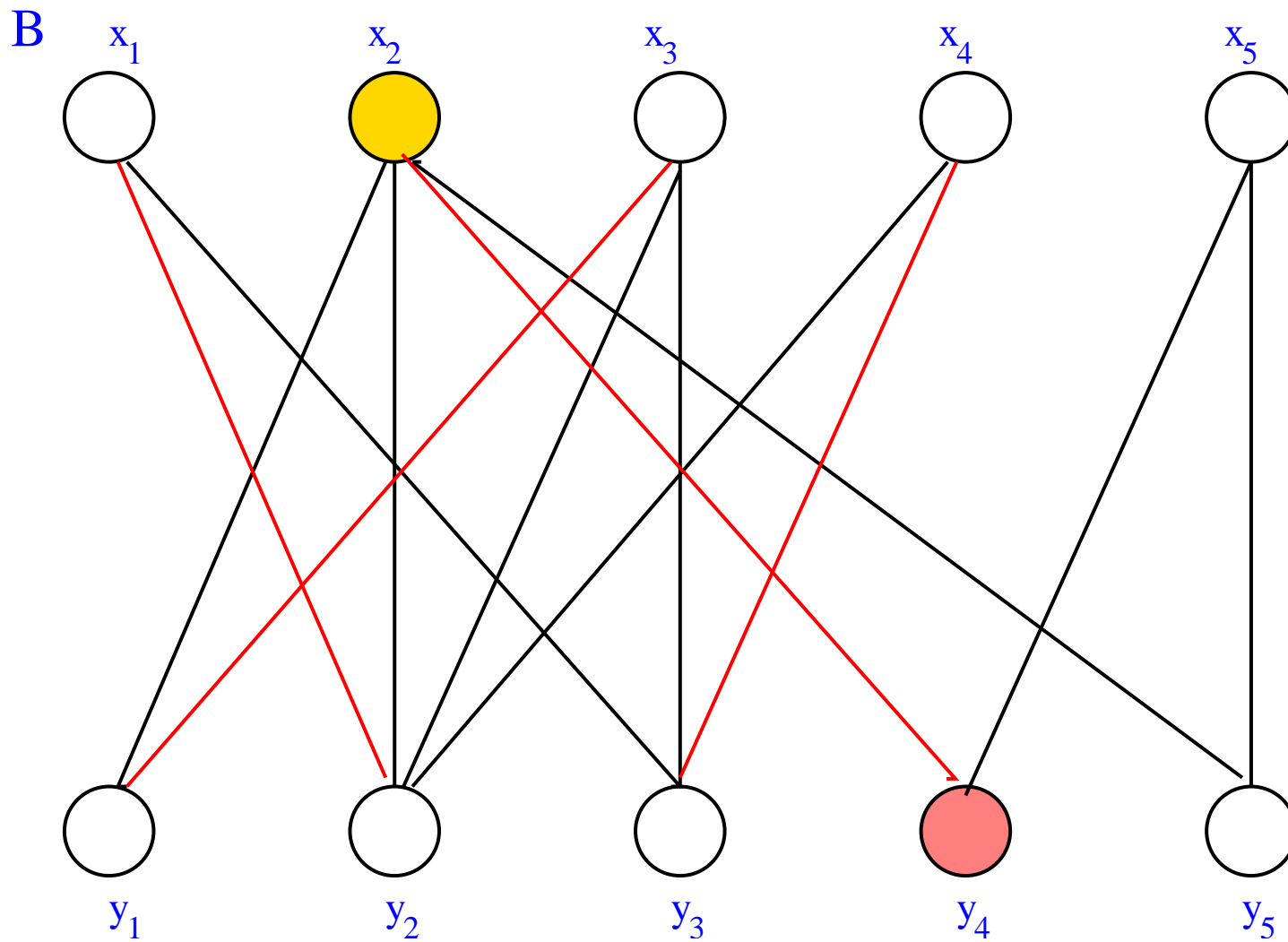
Przykład



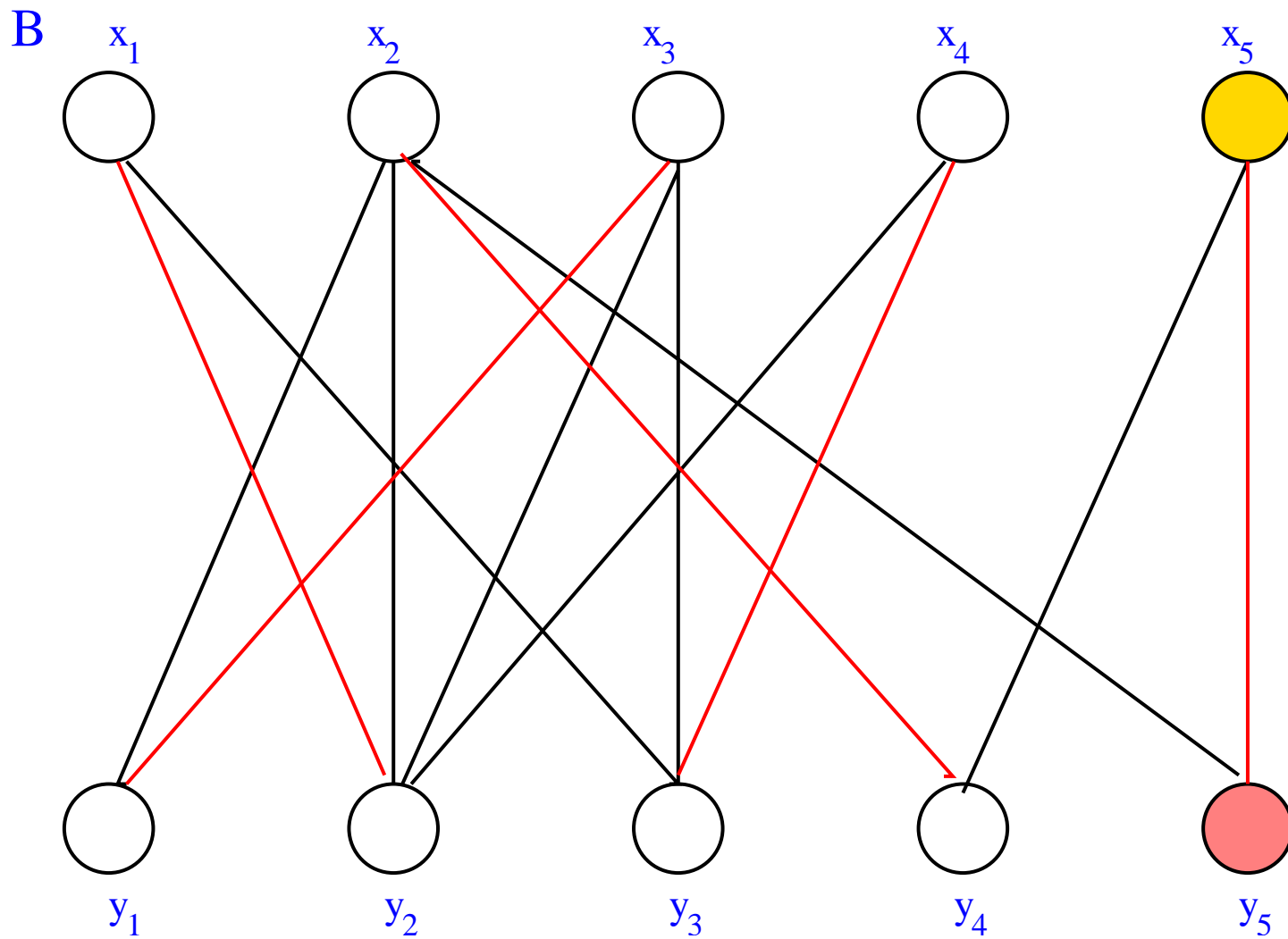
Przykład



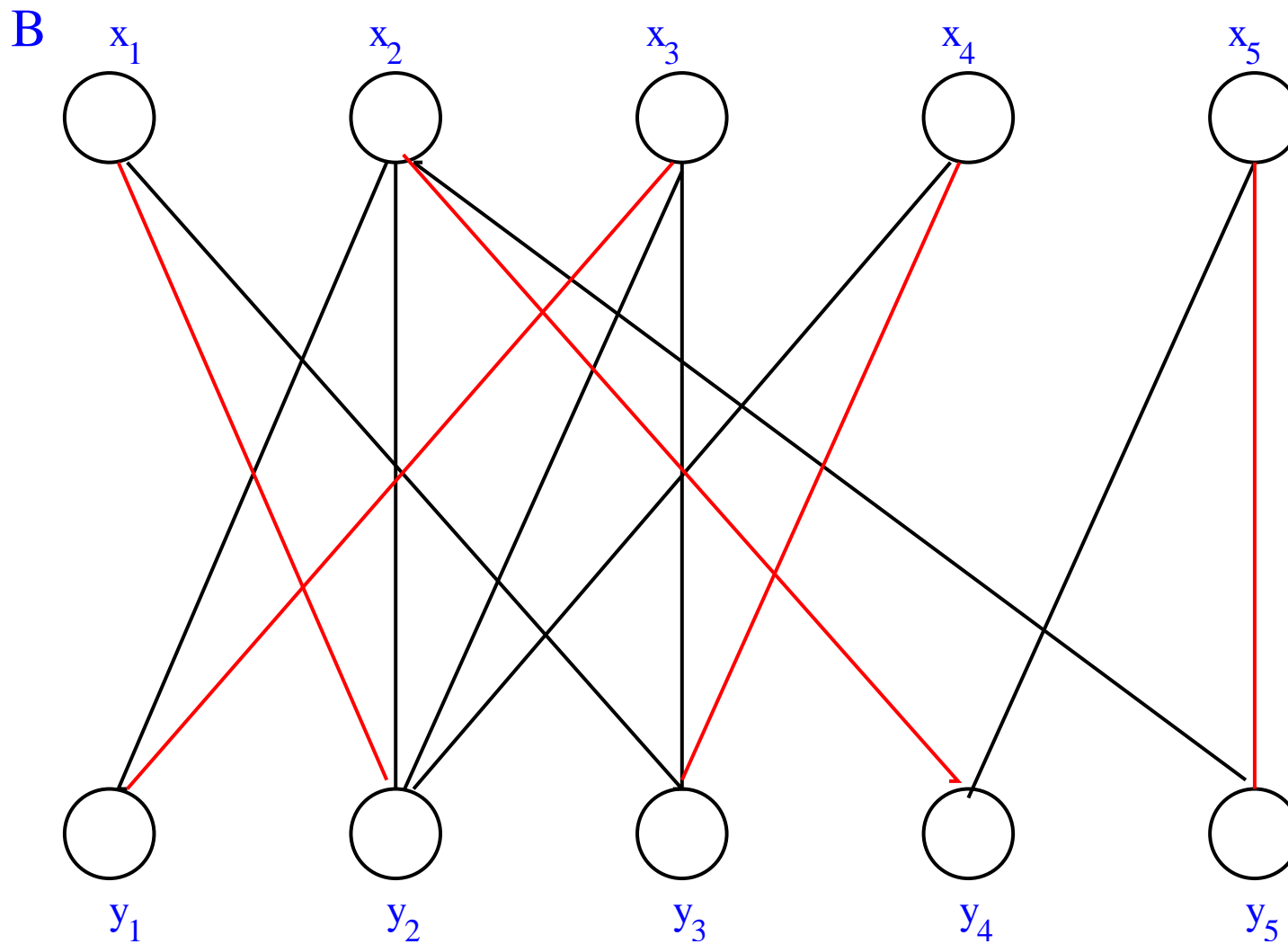
Przykład



Przykład



Przykład



7.3 Skojarzenia w ważonych grafach dwudzielnych

Niech będzie dany graf dwudzielny $G = (V, E)$, gdzie $V = (X, Y)$, oraz niech $w : E \rightarrow \mathbb{R}_+$ przyporządkowuje nieujemne wagi rzeczywiste krawędziom grafu G .

Problem: znajdź w grafie dwudzielnym G skojarzenie M o maksymalnej wadze, gdzie $w(M) = \sum_{e \in M} w(e)$

Wprowadźmy funkcję l określoną na zbiorze wierzchołków $V = X \cup Y$: $l : V \rightarrow \mathbb{R}_+$ taką, że dla każdego $x \in X$ i dla każdego $y \in Y$

$$l(x) + l(y) \geq w(xy), \quad (1)$$

gdzie $w(xy)$ jest wagą krawędzi xy .

Przykładem funkcji l spełniającej warunek (1) może być funkcja :

$$l(x) := \max_{y \in Y} w(xy) \quad \forall x \in X \quad \text{oraz} \quad l(y) := 0 \quad \forall y \in Y$$

Twierdzenie (Egerváry, 1931). *Maksymalna waga skojarzenia w $G = (V, E)$ jest równa $l^*(V) = \sum_{v \in V} l^*(v)$, gdzie $l^*(V) = \min_{l \in \mathcal{L}} l(V)$, natomiast \mathcal{L} oznacza klasę funkcji spełniających warunek (1).*

Dowód. (szkic) Niech M będzie dowolnym skojarzeniem w G i niech $l \in \mathcal{L}$. Wtedy

$$w(M) = \sum_{e \in M} w(e) = \sum_{e=xy \in M} (l(x) + l(y)) \leq \sum_{v \in V} l(v). \quad (2)$$

Aby pokazać, że w (2) zachodzi równość, przyjmijmy, że l^* jest funkcją z \mathcal{L} osiągającą minimalną wartość $l^*(V)$. Oznaczmy przez F zbiór krawędzi grafu G dla których w (1) mamy równość, a przez R zbiór wierzchołków v dla których $l^*(v) > 0$.

Jeżeli F zawiera skojarzenie M^* “pokrywające” R , to wtedy

$$w(M^*) = \sum_{e \in M^*} w(e) = \sum_{v \in V} l^*(v) = l^*(V).$$

Jeżeli F nie zawiera skojarzenia M^* “pokrywającego” R , to można pokazać, że musi istnieć zbiór niezależny $S \subseteq R$ taki, że $|N(S)| < |S|$. Wtedy $\exists \alpha > 0$ takie, że $l(v) \leftarrow l(v) - \alpha$ dla $v \in S$, $l(v) \leftarrow l(v) + \alpha$ gdy $v \in N(S)$, otrzymujemy “mniejsze” l , czyli sprzeczność! ■

Algorytm węgierski

Dane: ważony graf dwudzielny G ($V = (X, Y)$) oraz skojarzenie M

Poszukiwane: skojarzenie M takie, że $w(M)$ jest największa

1. Zorientuj każdą krawędź $e = xy$, gdzie $x \in X$, $y \in Y$ grafu G w następujący sposób:
 - (a) jeżeli $e \in M$ to zorientuj e od y do x i przyjmij $w(e)$ za długość łuku
 - (b) jeżeli $e \notin M$ to zorientuj e od x do y i przyjmij $-w(e)$ za długość łuku
2. Znajdź zbiory X_M i Y_M wierzchołków M -nienasyconych w zbiorach, odpowiednio, X i Y .

3 Znajdź **najkrótszą** ścieżkę M -powiększającą P , poprzez wyznaczenie najkrótszej skierowanej ścieżki z X_M do Y_M , $M \leftarrow M \div E(P)$. Jeżeli znaleziona najkrótsza ścieżka ma ujemną długość to przejdź do kroku 1, w przeciwnym razie STOP.

Twierdzenie . Skojarzenie w ważonym grafie dwudzielnym znalezione za pomocą algorytmu węgierskiego jest ekstremalne.

Twierdzenie . Algorytm węgierski znajduje skojarzenie o największej wadze w ważonym grafie dwudzielnym w czasie $O(n^2m)$

Dowód. Zauważmy, że w powyższym algorytmie mamy co najwyżej n iteracji, z których każda może być wykonana za pomocą algorytmu Bellmana-Forda (znajdowanie najkrótszej ścieżki z X_M do Y_M), w czasie $O(nm)$.

Algorithm 0.0.1: METODA WĘGIERSKA($G = ((X, Y), E)$)

$M \leftarrow \emptyset$

repeat

$X_M \leftarrow Y_M \leftarrow \emptyset$

for each $e = xy \in E$

do $\begin{cases} X_M \leftarrow X_M \cup \{x\}; & Y_M \leftarrow Y_M \cup \{y\} \\ \textbf{if } e \in M \\ \quad \textbf{then } e \leftarrow \overrightarrow{yx}; & w(\overrightarrow{e}) \leftarrow w(e) \\ \quad \textbf{else } e \leftarrow \overrightarrow{xy}; & w(\overrightarrow{e}) \leftarrow -w(e) \end{cases}$

$X_M \leftarrow X \setminus X_M; \quad Y_M \leftarrow Y \setminus Y_M$

$exists \leftarrow$ czy istnieje skierowana ścieżka z X_M do Y_M

if $exists$

then $\begin{cases} \overrightarrow{P} \leftarrow \text{najkrótsza skierowana ścieżka z } X_M \text{ do } Y_M \\ M \leftarrow M \div E(\overrightarrow{P}) \end{cases}$

until not $exists$ **or** $w(\overrightarrow{P}) < 0$

return (M)

Problem optymalnego przydziału zadań

Oznaczmy przez E_l zbiór krawędzi

$$E_l = \{xy \in E(G) : l(x) + l(y) = w(xy)\}$$

a przez G_l odpowiedni rozpięty podgraf grafu G .

Dane: graf dwudzielny $G = (V, E)$, $V = (X, Y)$ z wagami na krawędziach, funkcja etykietująca wierzchołki l , graf G_l oraz jego skojarzenie M

Poszukiwane: skojarzenie o maksymalnej wadze nasycające zbiór wierzchołków X (skojarzenie optymalne)

Algorytm Kuhna - Munkresa - (Edmondsa)

1. Jeżeli M nasyca wszystkie wierzchołki X , to M jest skojarzeniem optymalnym - STOP. W przeciwnym razie niech u , ($u \in X$) będzie M -nienasyconym wierzchołkiem, $S := \{u\}$, $T := \emptyset$
2. Jeżeli $T \subset N_{G_l}(S)$, to przejdź do kroku 3. Jeżeli $T = N_{G_l}(S)$, to obliczamy

$$\alpha = \min_{x \in S, y \in Y - T} \{l(x) + l(y) - w(xy)\} \quad (\alpha > 0)$$

$l^*(v) \leftarrow l(v) - \alpha$, dla $v \in S$; $l^*(v) \leftarrow l(v) + \alpha$, dla $v \in T$;
 $l^*(v) \leftarrow l(v)$ dla wszystkich v poza S i poza T ; $G_l \leftarrow G_{l^*}$

3 Wybieramy $y \in N_{G_l}(S) - T$. Jeżeli y jest M -nasycony, $yz \in M$, to $S \leftarrow S \cup \{z\}$, $T \leftarrow T \cup \{y\}$ i przejdź do kroku 2. W przeciwnym razie mamy M -powiększającą ścieżkę P z u do y w G_l ; $M \leftarrow M \div E(P)$ i przejdź do kroku 1.

Przykład . Wyznaczyć optymalne skojarzenie (skojarzenie doskonałe o maksymalnej wadze) w grafie o macierzy wag:

$$\begin{array}{c}
 \\
 y_1 \\
 y_2 \\
 y_3 \\
 y_4 \\
 y_5
 \end{array}
 \begin{pmatrix}
 x_1 & x_2 & x_3 & x_4 & x_5 \\
 3 & 5 & 5 & 4 & 1 \\
 2 & 2 & 0 & 2 & 2 \\
 2 & 4 & 4 & 1 & 0 \\
 0 & 1 & 1 & 0 & 0 \\
 1 & 2 & 1 & 3 & 3
 \end{pmatrix}$$

Algorithm 0.0.1: KUHNMUNKRES($G = ((X, Y), E)$)

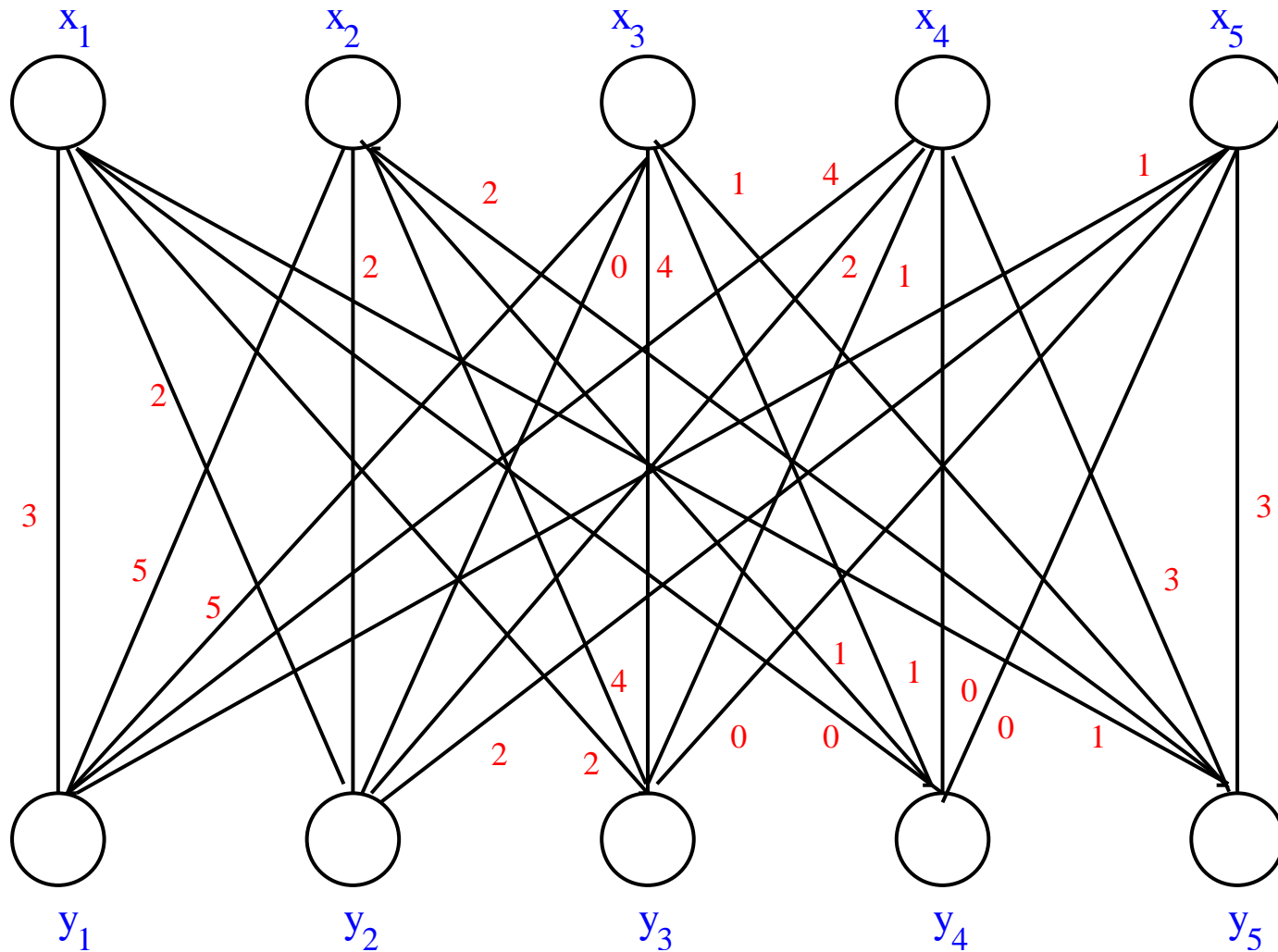
$M \leftarrow \emptyset$

while $|M| \neq |X|$

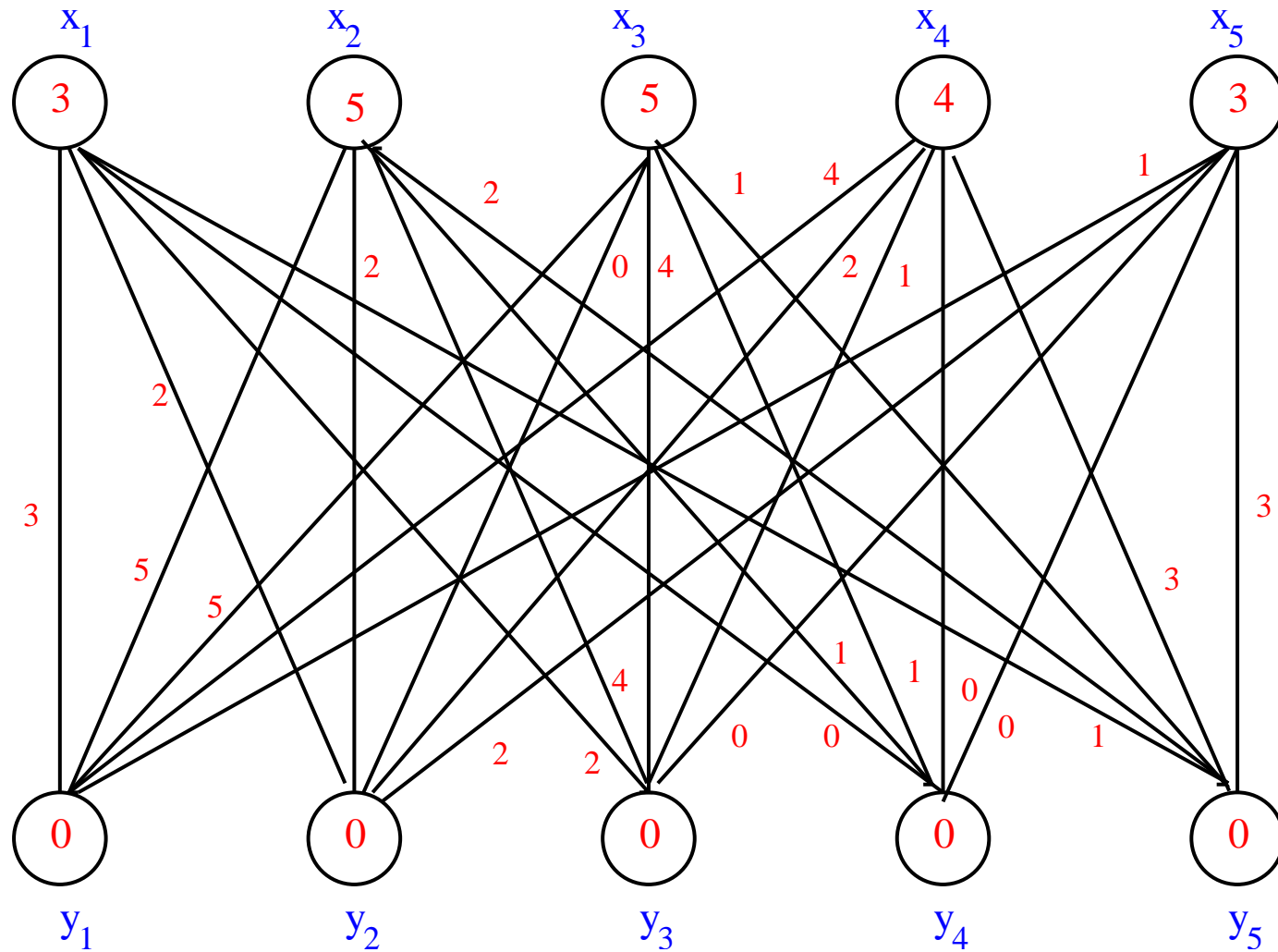
do {
 $u \leftarrow M$ -nienasycony wierzchołek z X
 $S \leftarrow \{u\}; T \leftarrow \emptyset$
 repeat
 if $N_G(S) = T$
 then $\left\{ \alpha \leftarrow \min_{x \in S, y \in Y - T} \{l(x) + l(y) - w(xy)\} \right.$
 for each $v \in S$
 do $l^*(v) \leftarrow l(v) - \alpha$
 for each $v \in T$
 do $l^*(v) \leftarrow l(v) + \alpha$
 for each $v \in V \setminus \{S \cup T\}$
 do $l^*(v) \leftarrow l(v)$
 $y \leftarrow$ dowolny element z $N_G(S) - T$
 if $y \in M$ -nasycone
 then $S \leftarrow S \cup \{z\}; T \leftarrow T \cup \{y\}$
 until $y \in M$ -nienasycone
 $P \leftarrow M$ -przemienna ścieżka powiększającą z u do y
 $M \leftarrow M \div E(P)$
}

return (M)

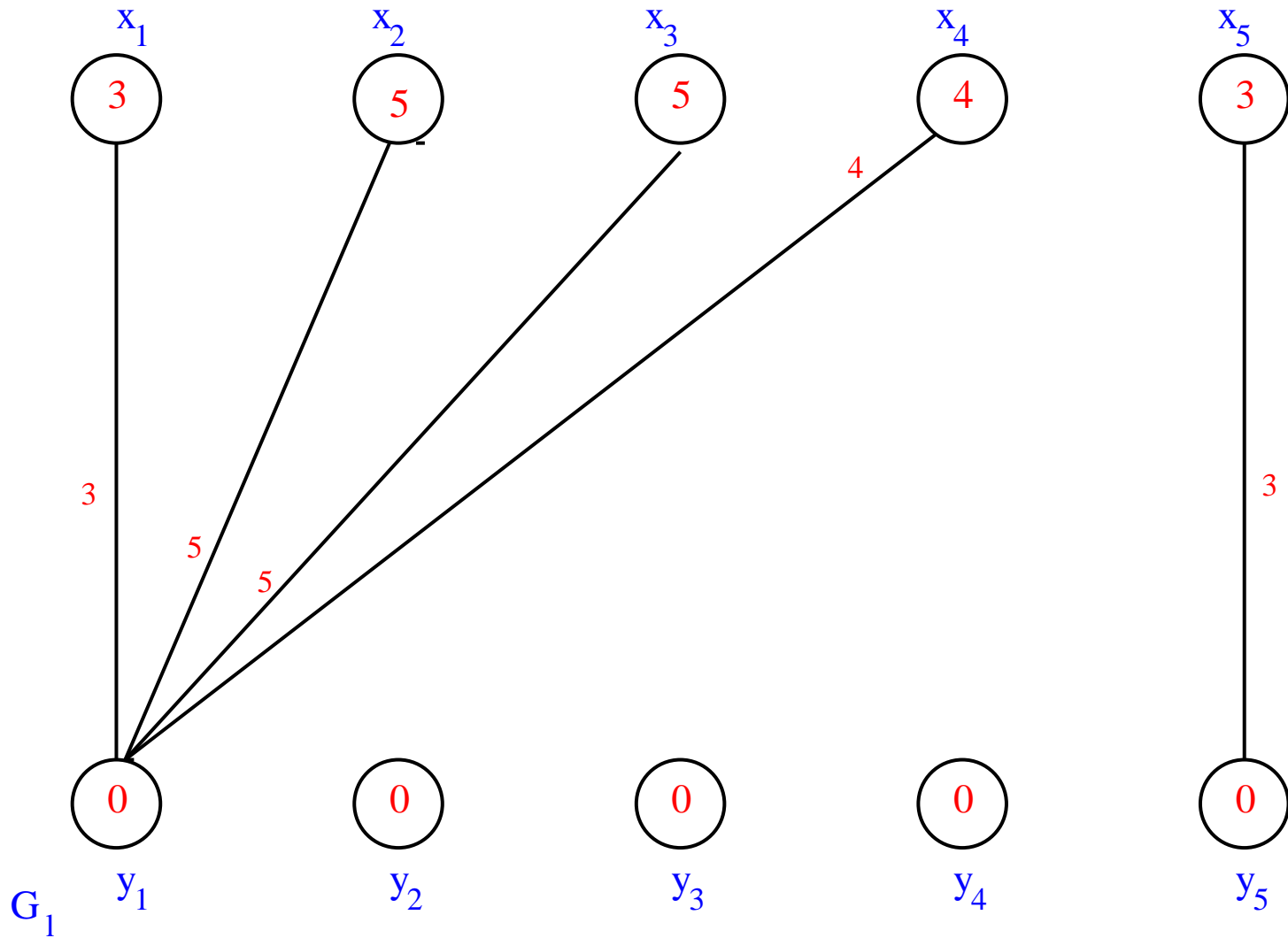
Przykład



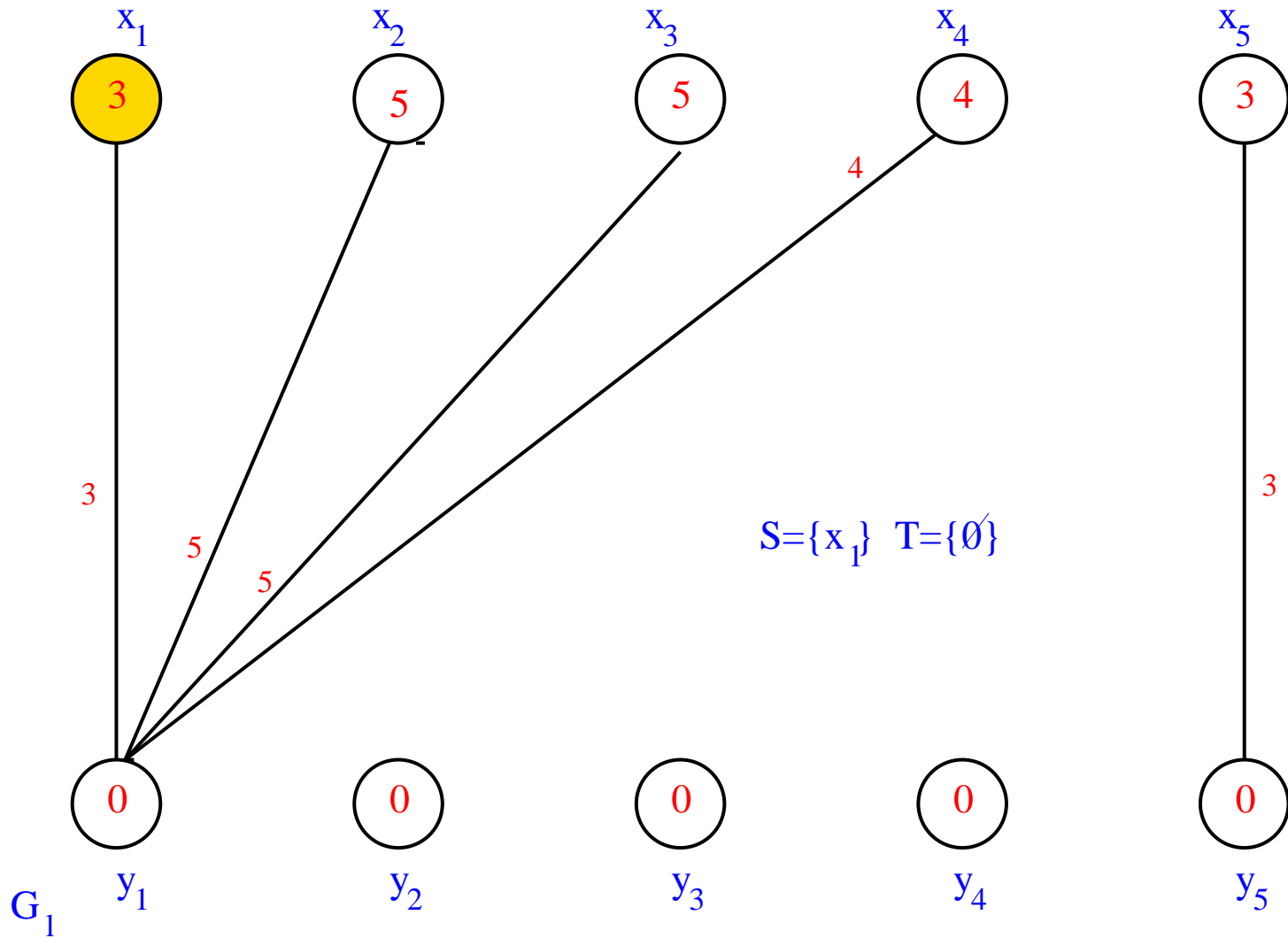
Przykład



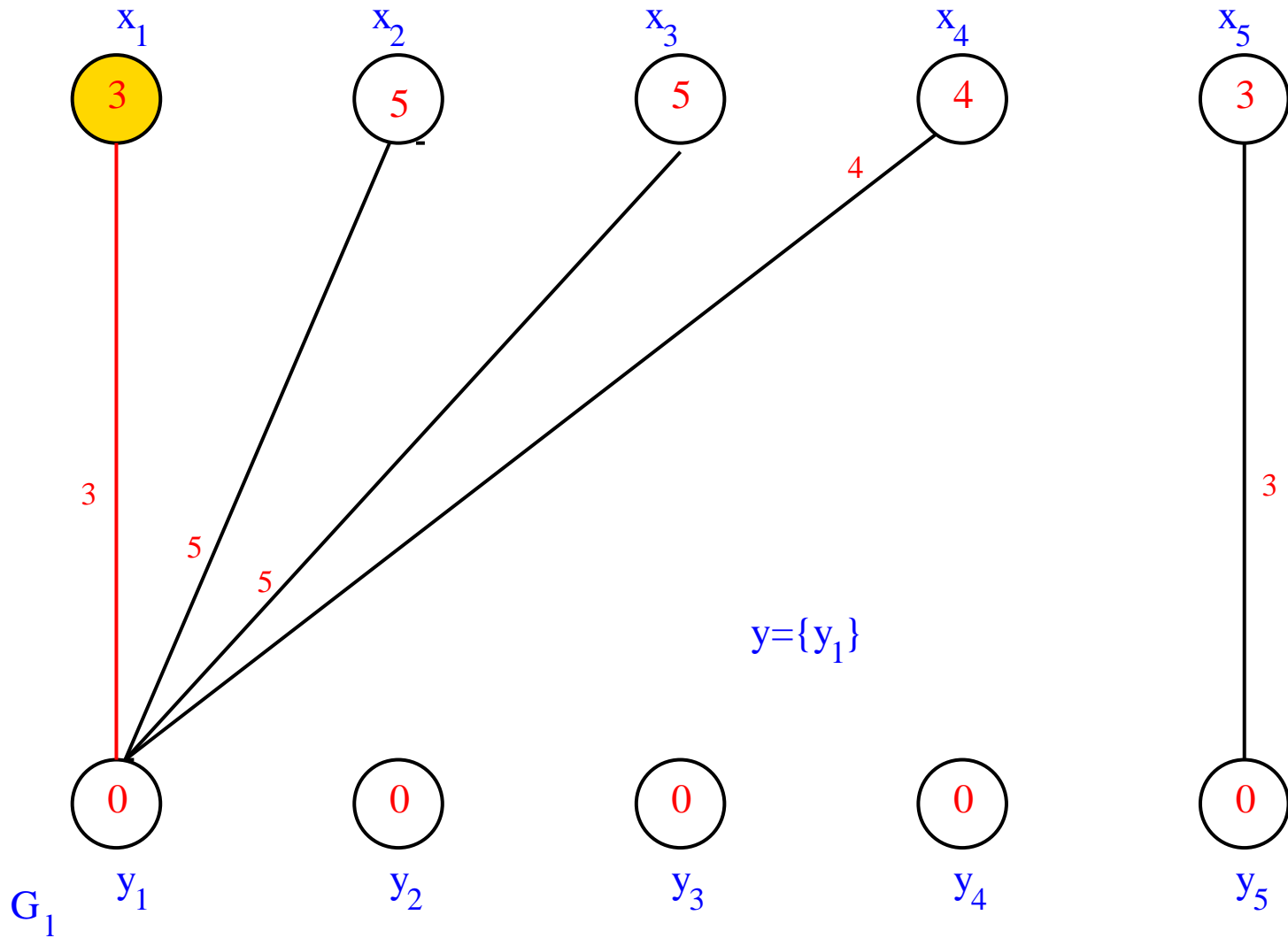
Przykład



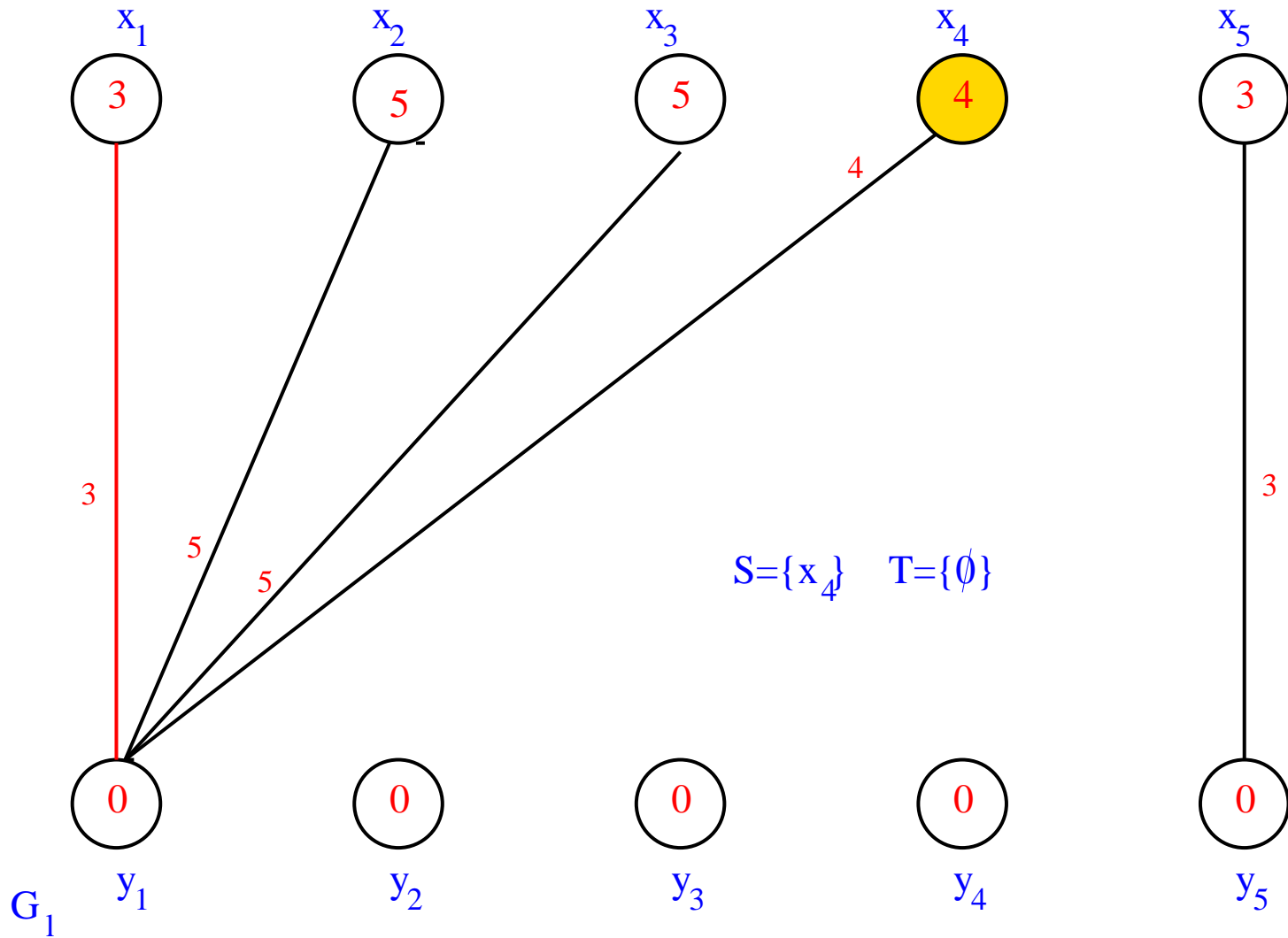
Przykład



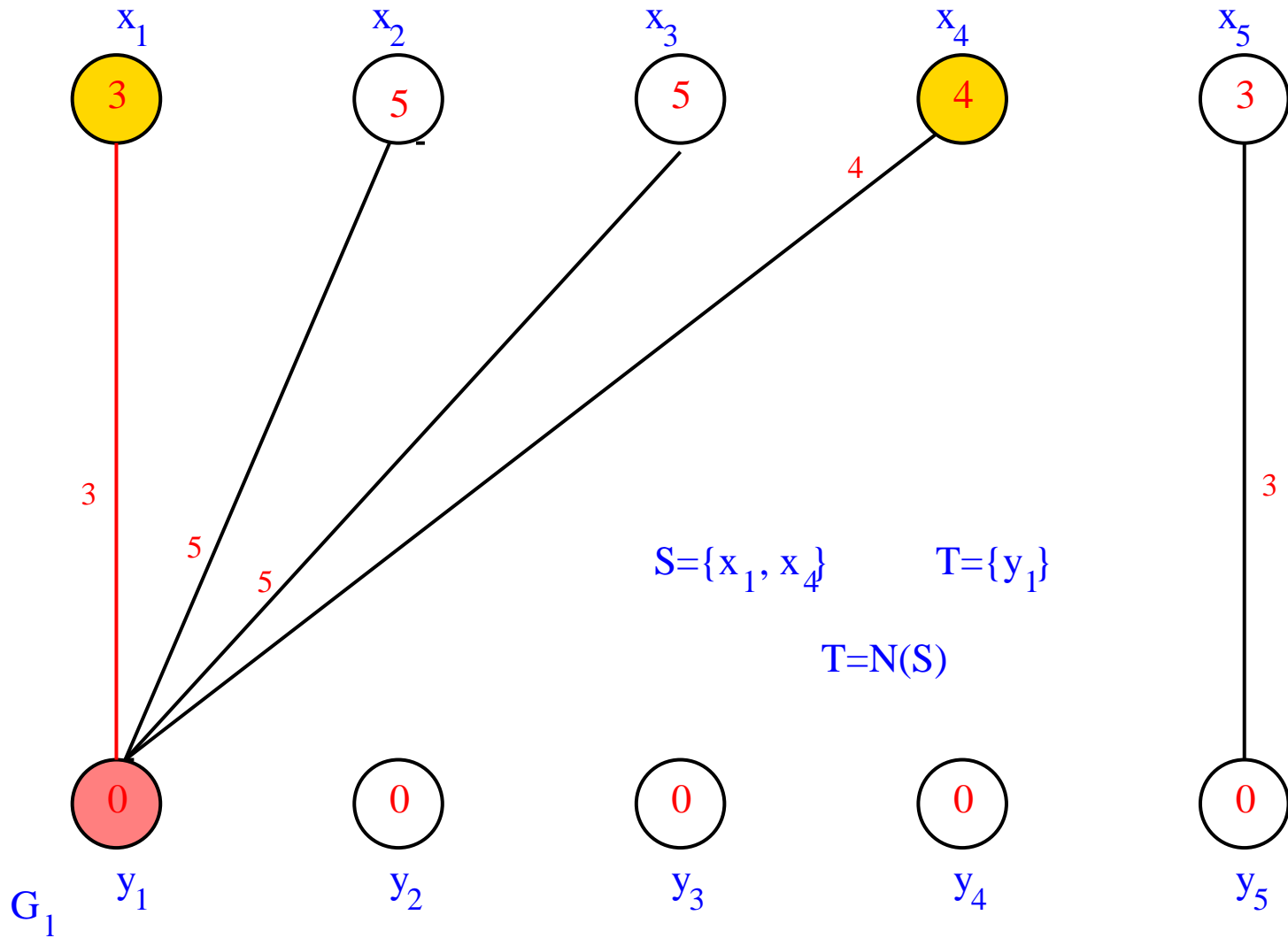
Przykład



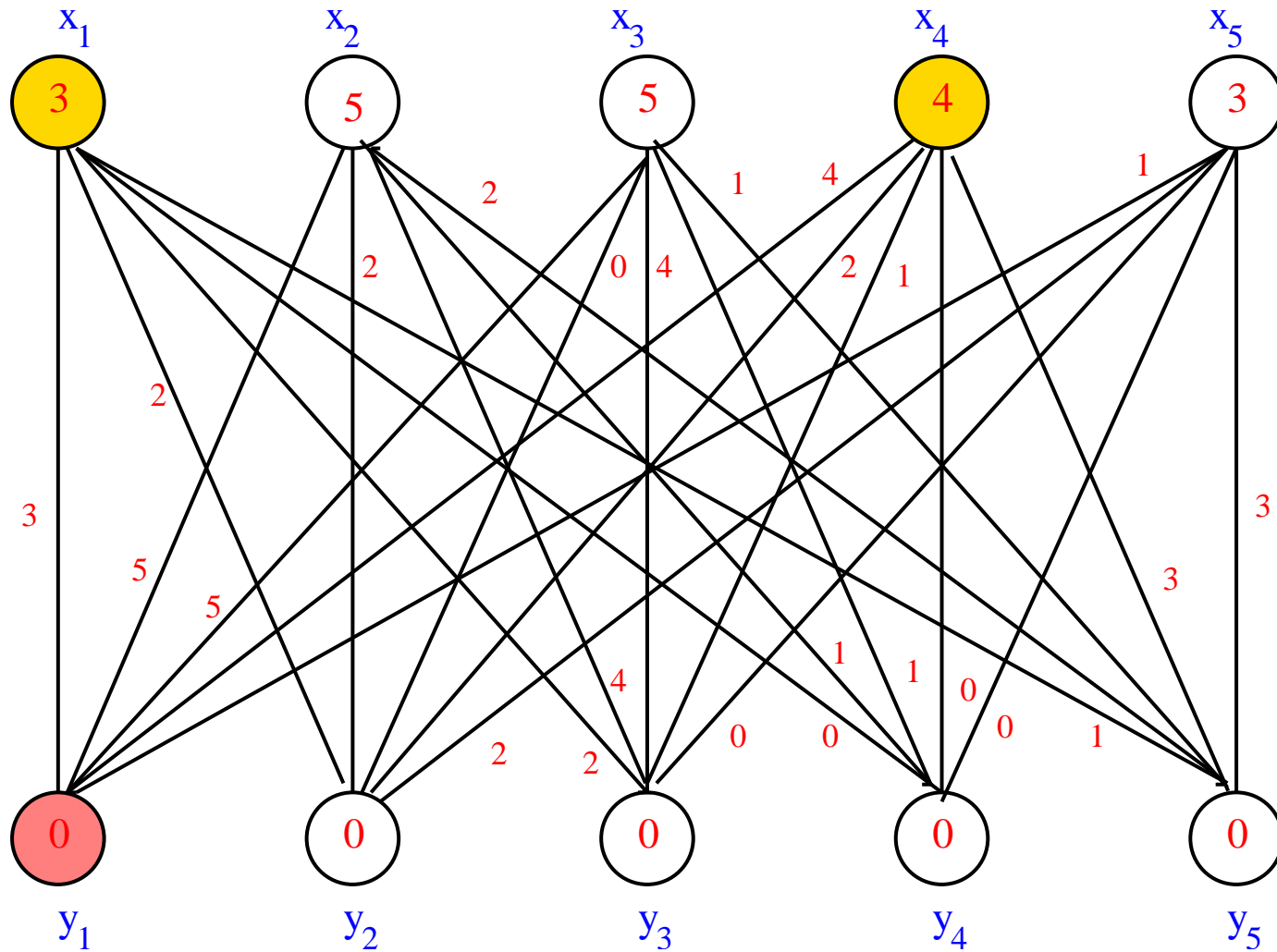
Przykład



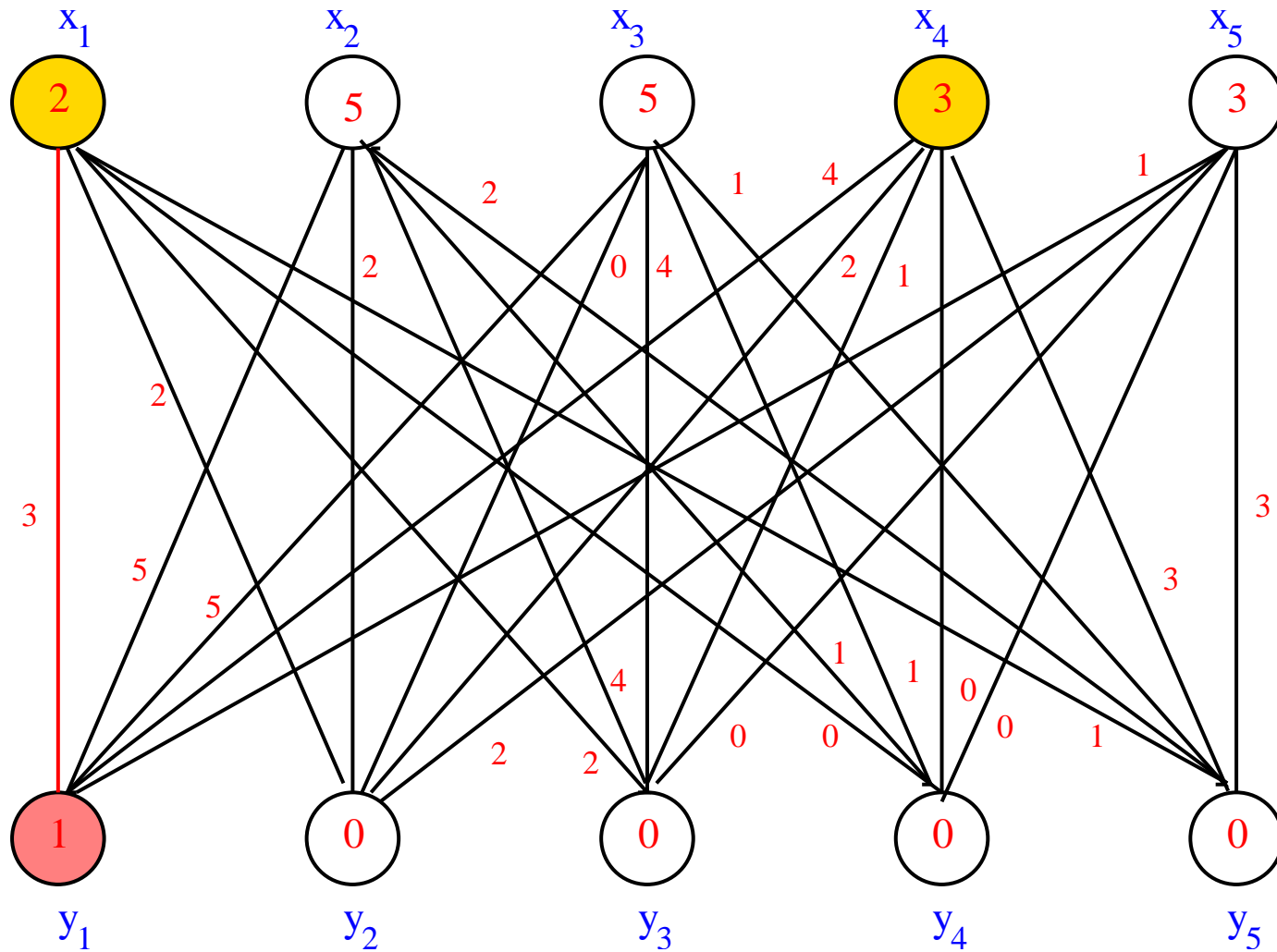
Przykład



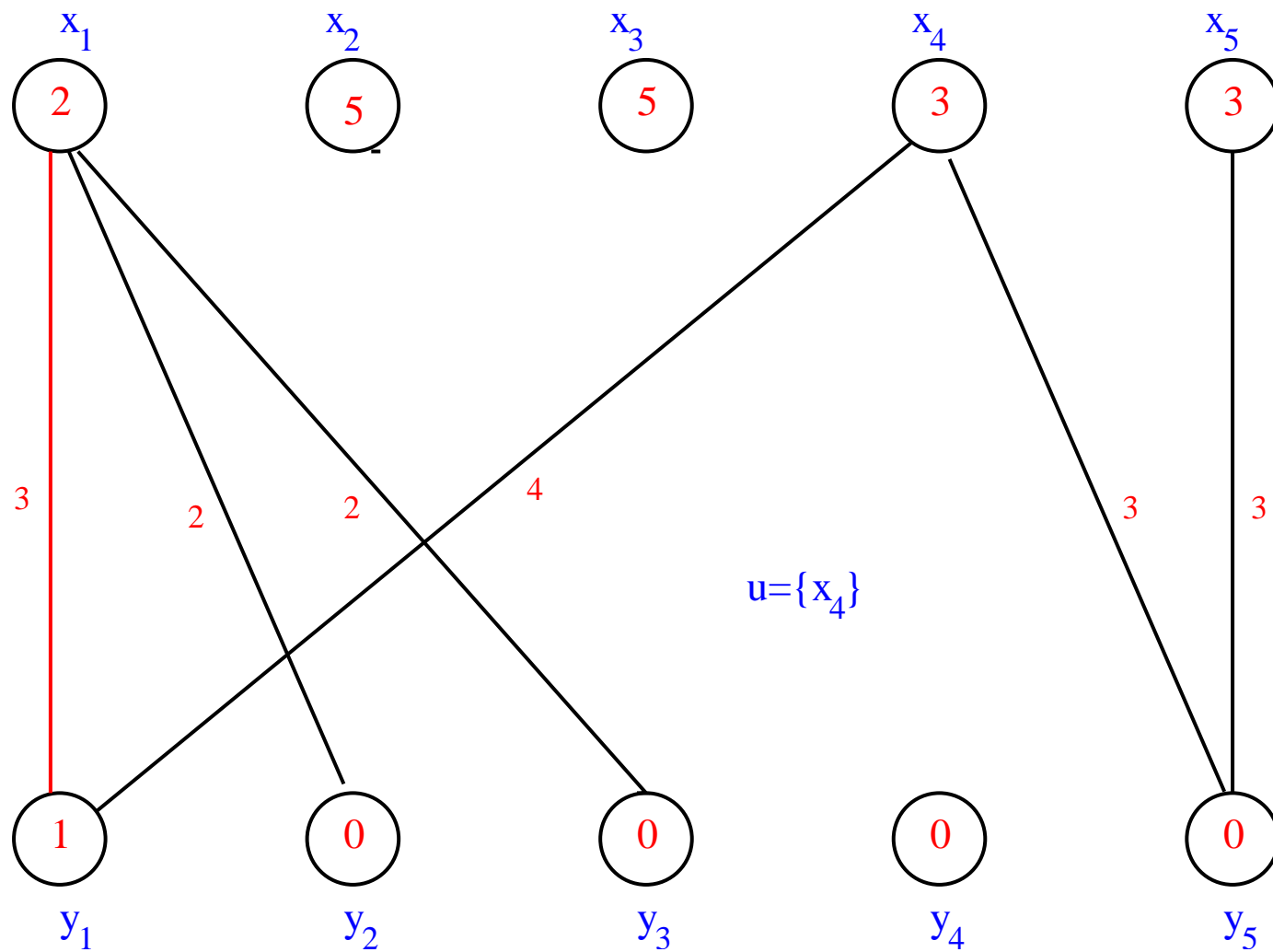
Przykład



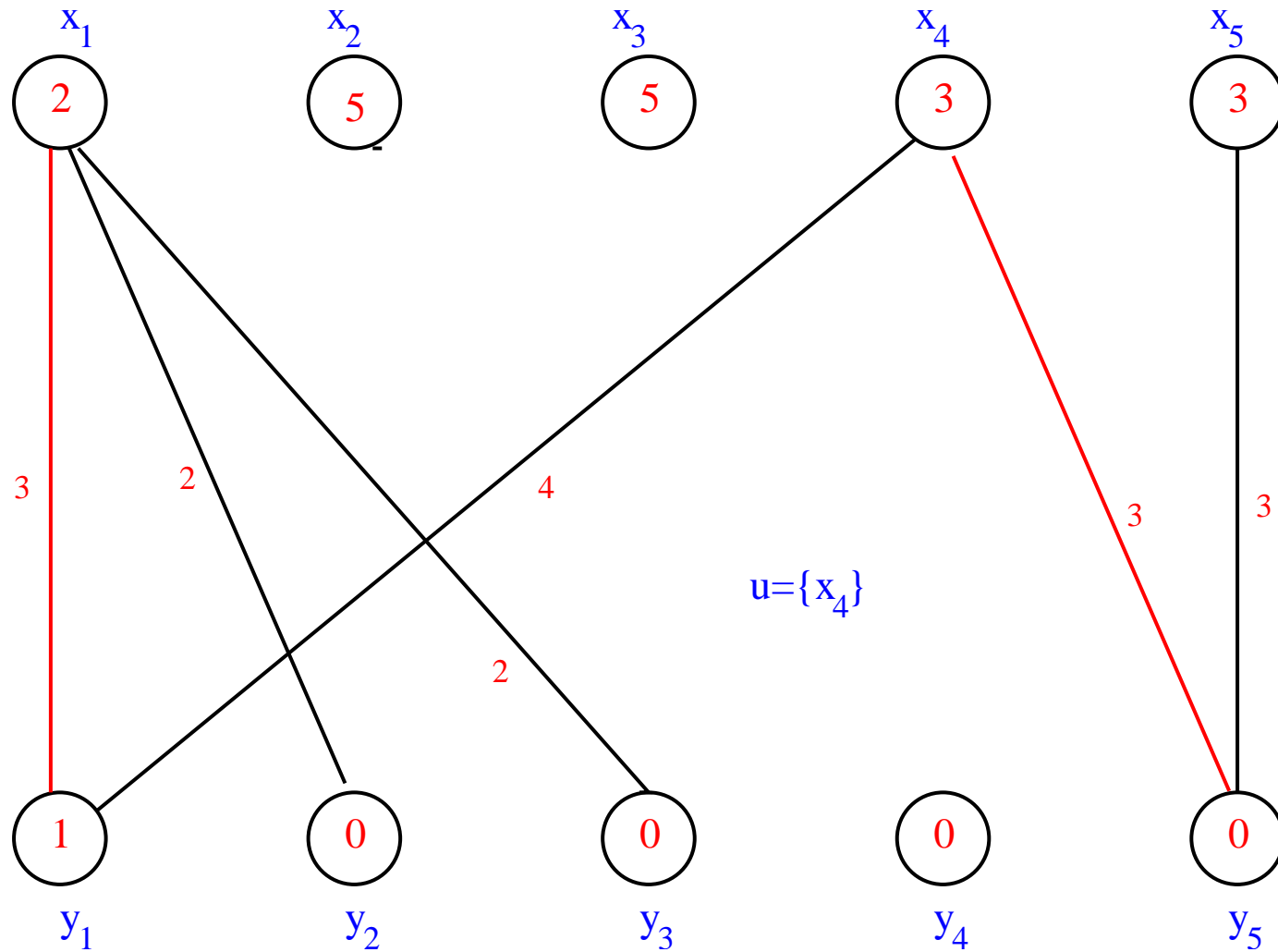
Przykład



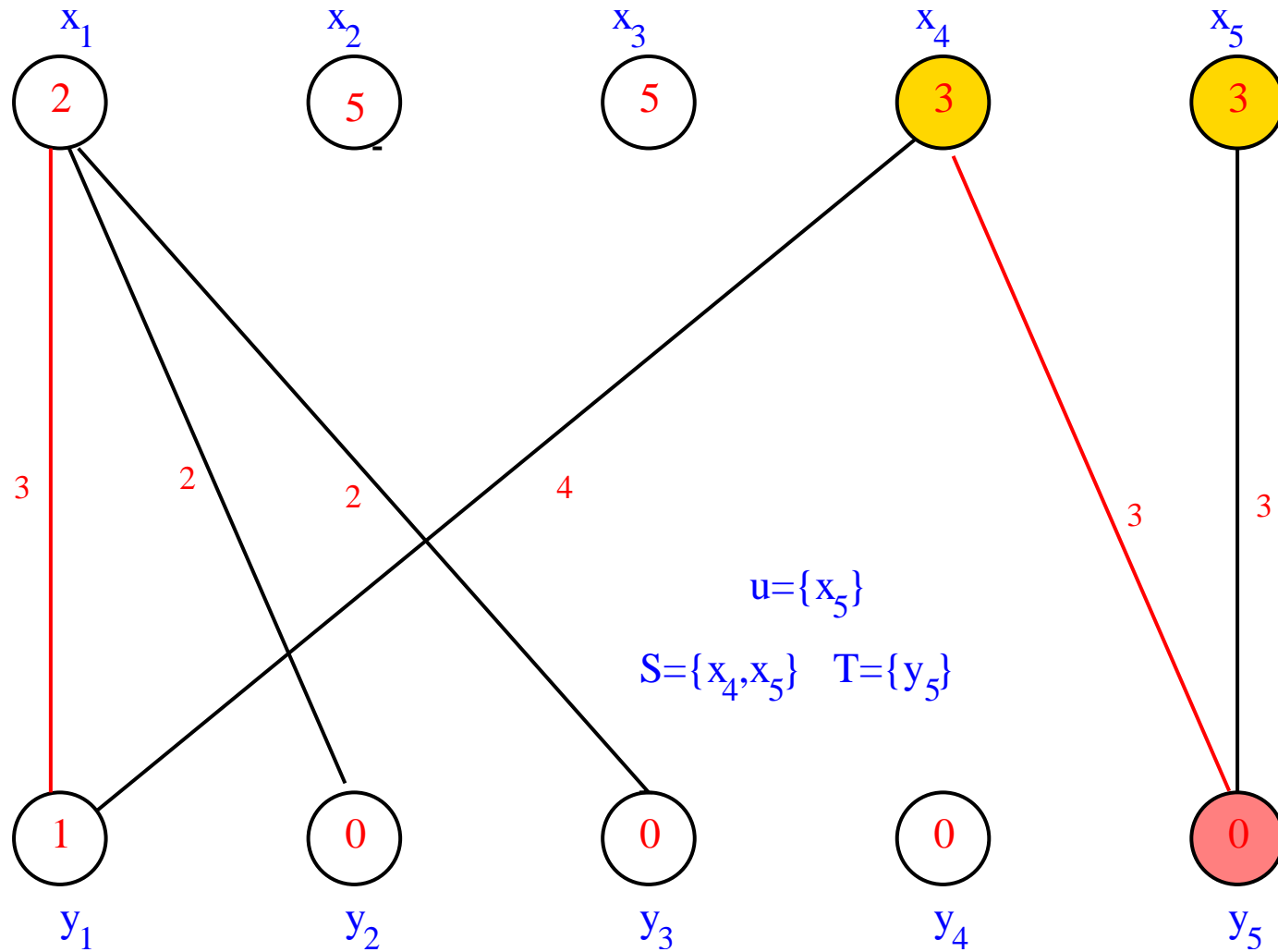
Przykład



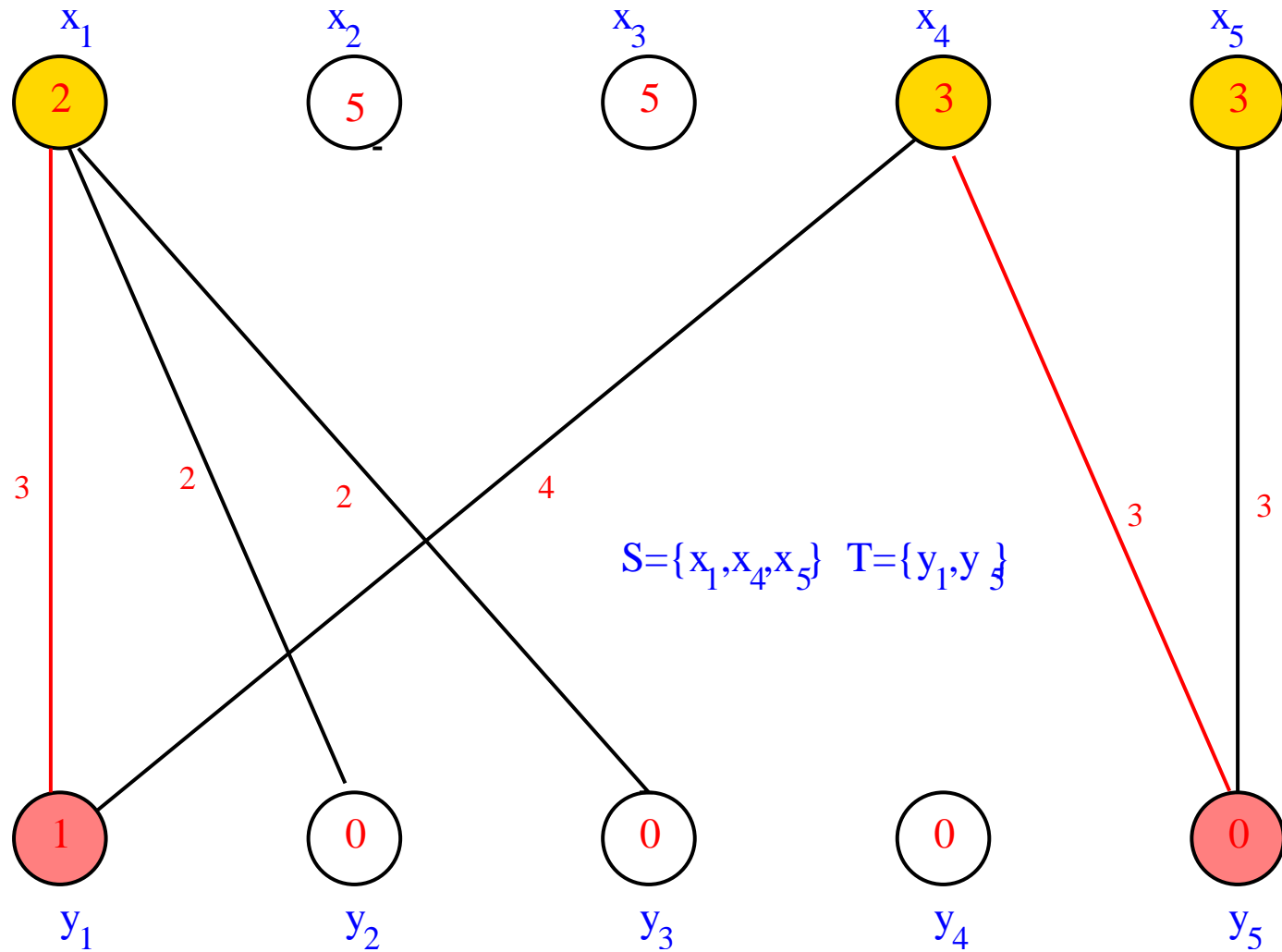
Przykład



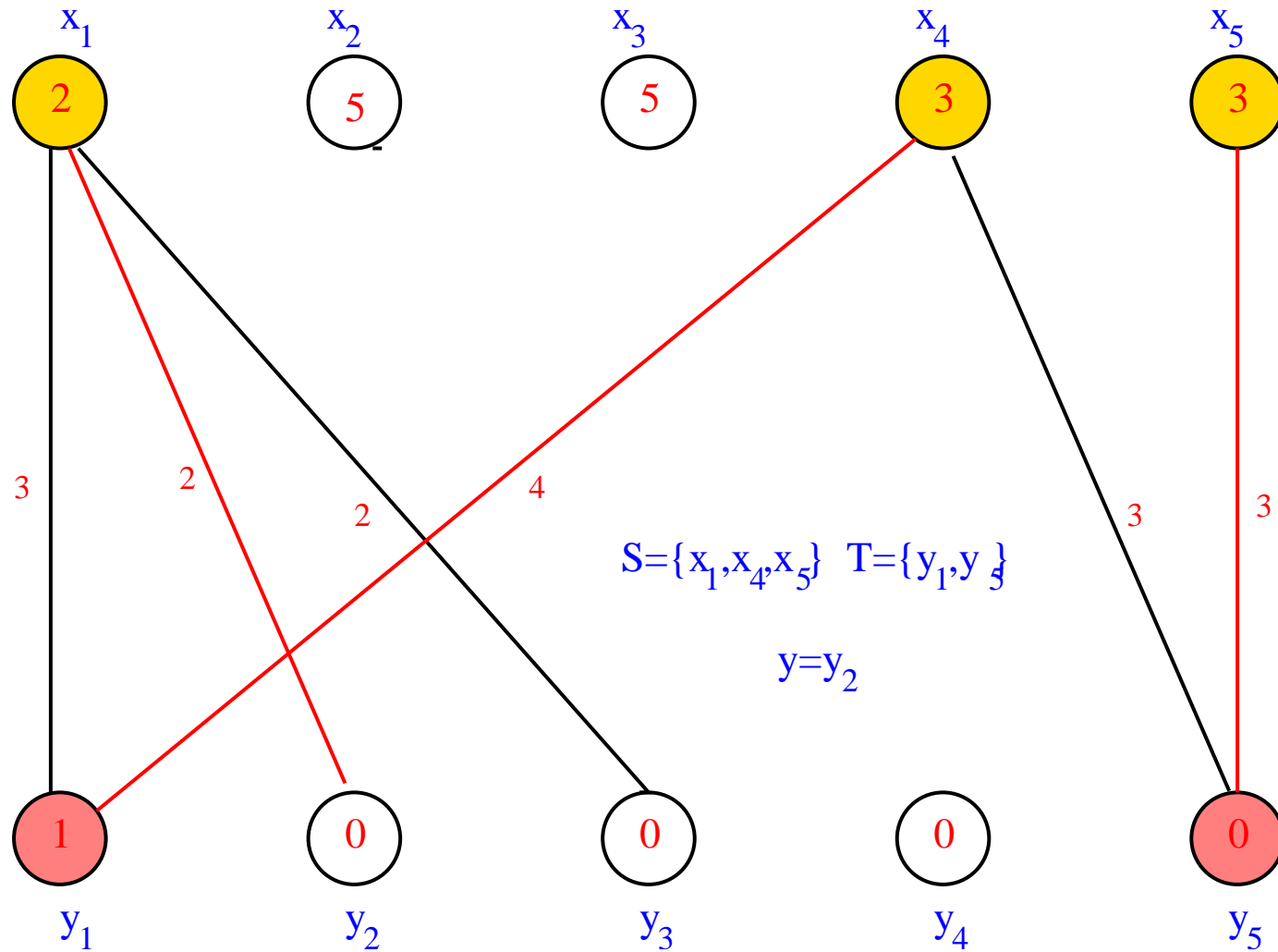
Przykład



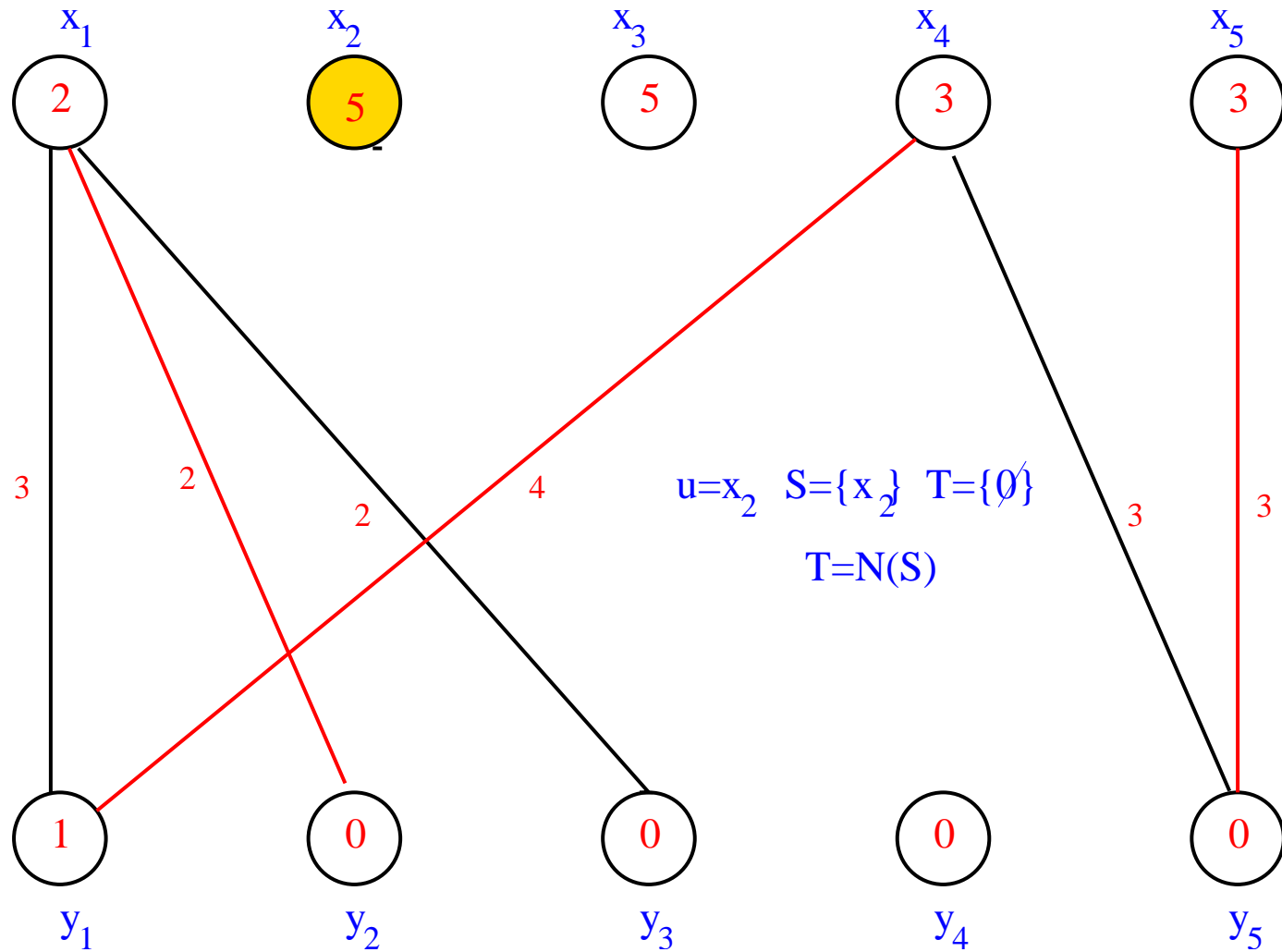
Przykład



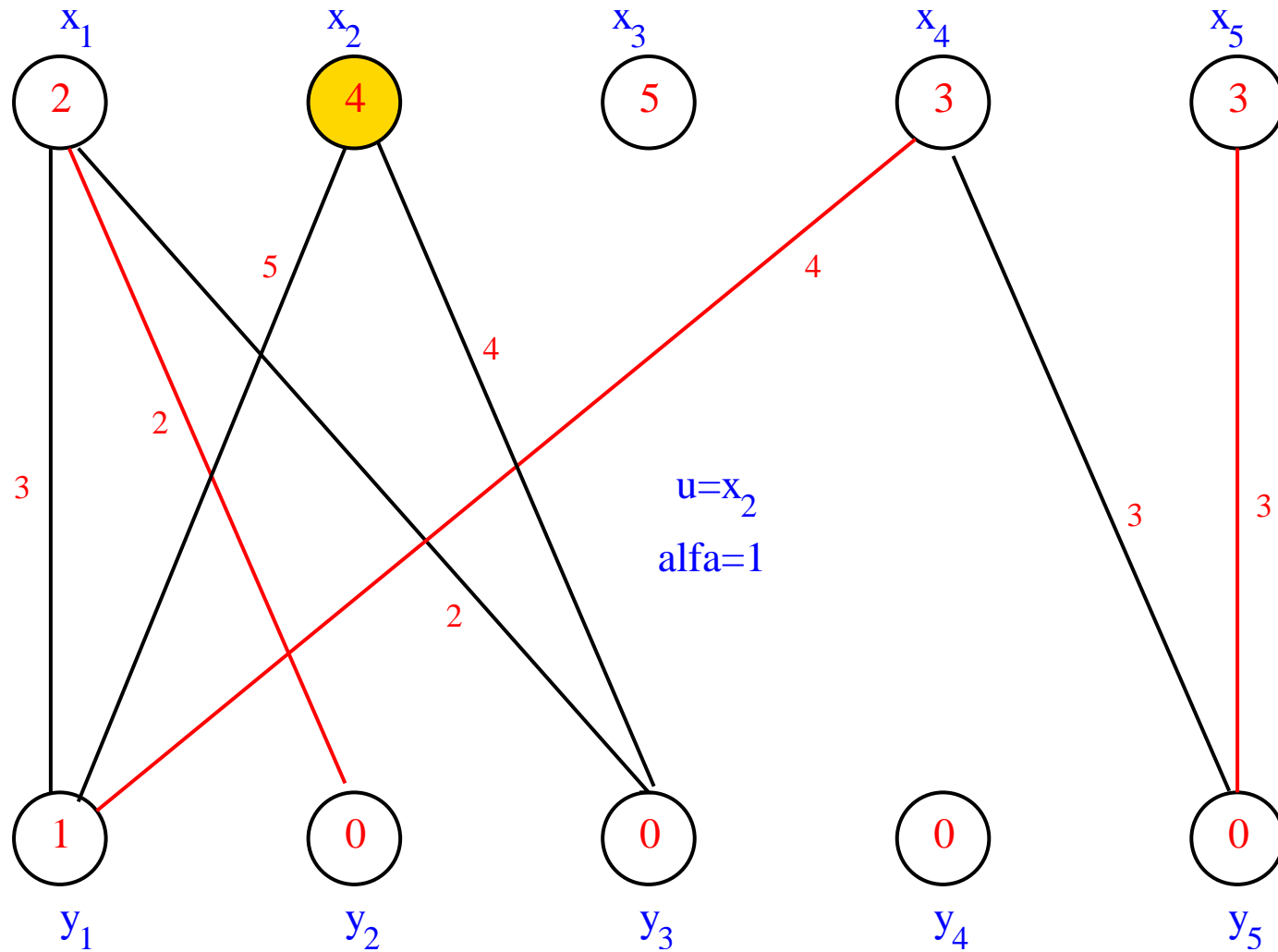
Przykład



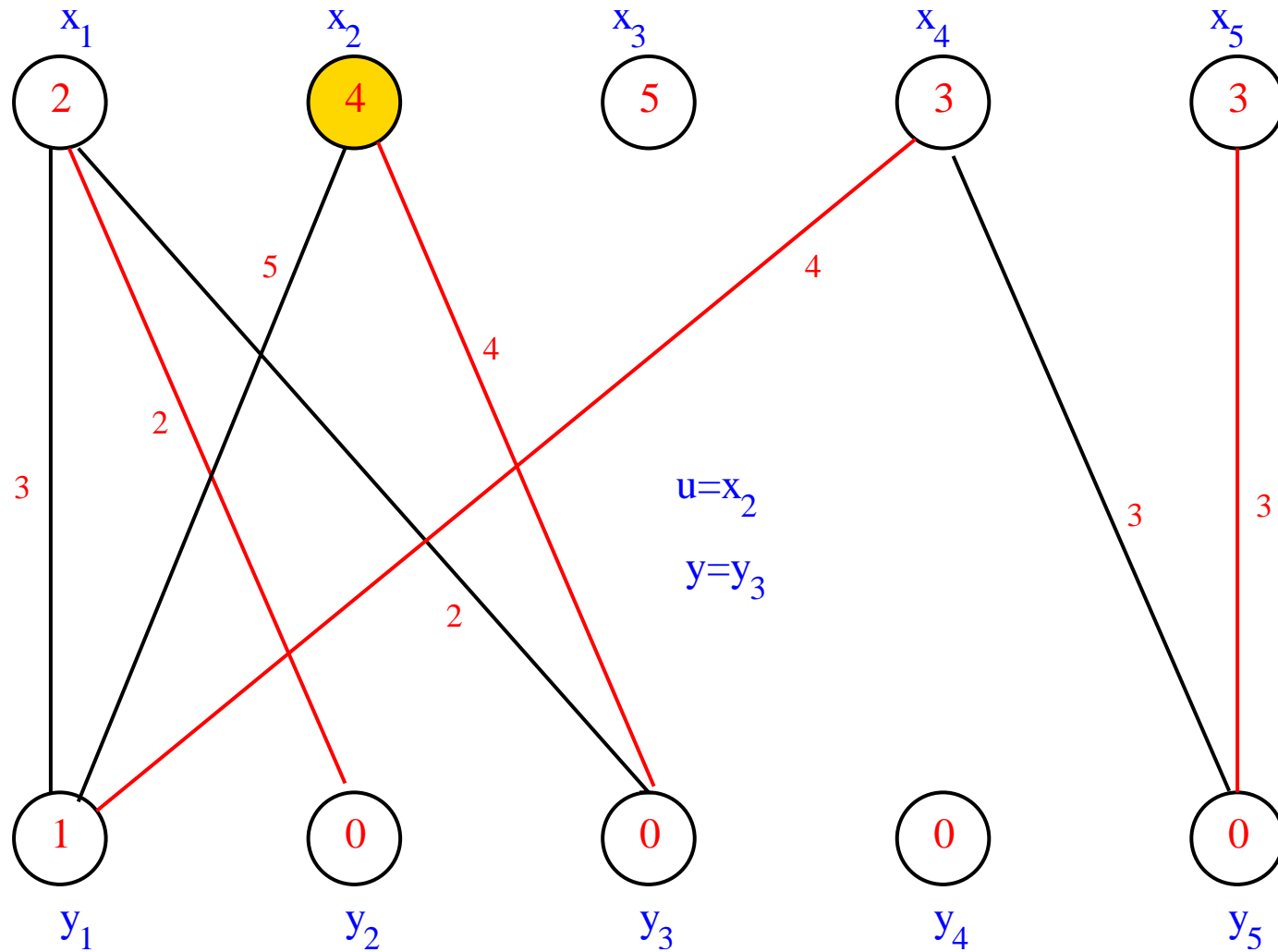
Przykład



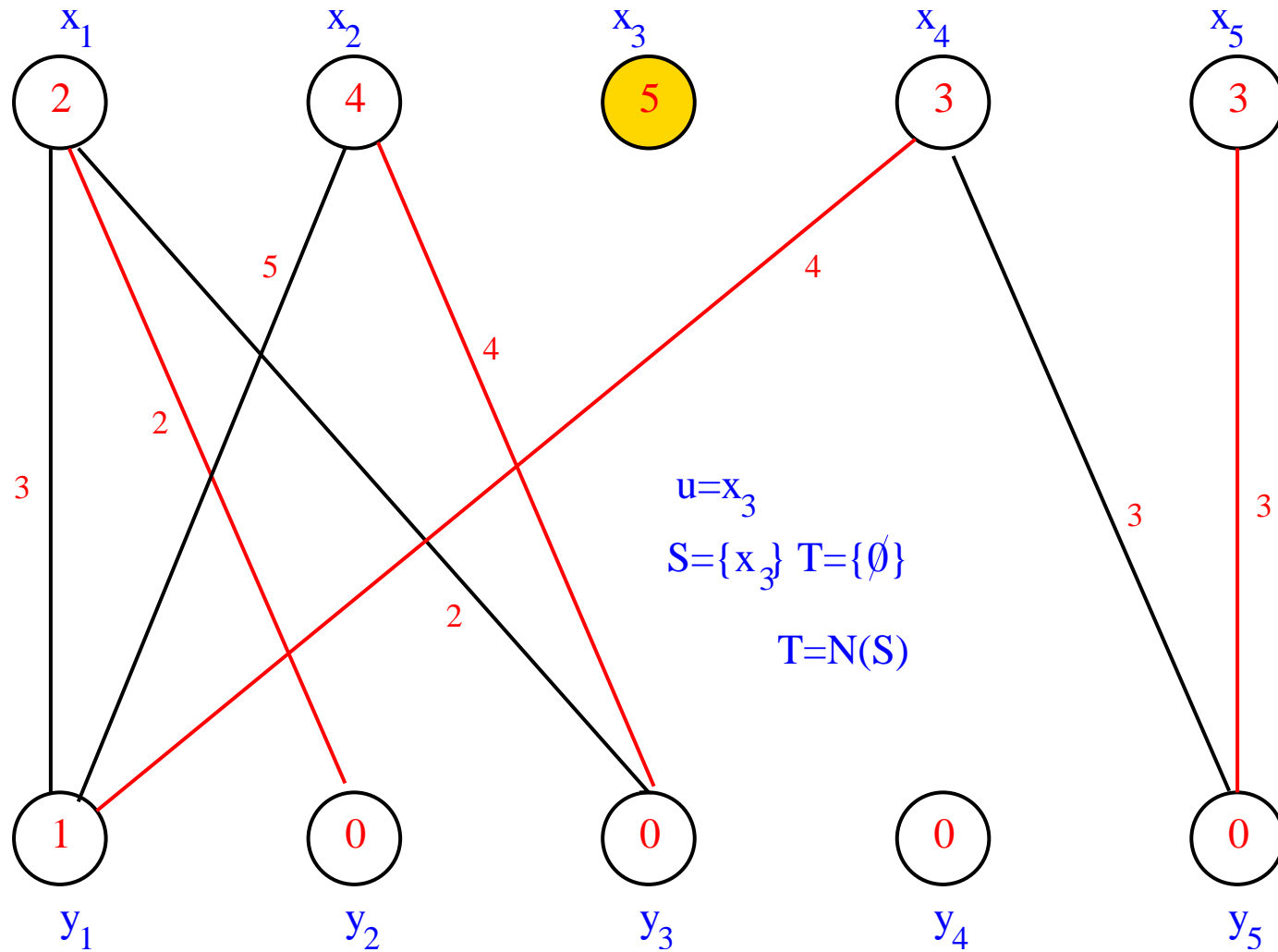
Przykład



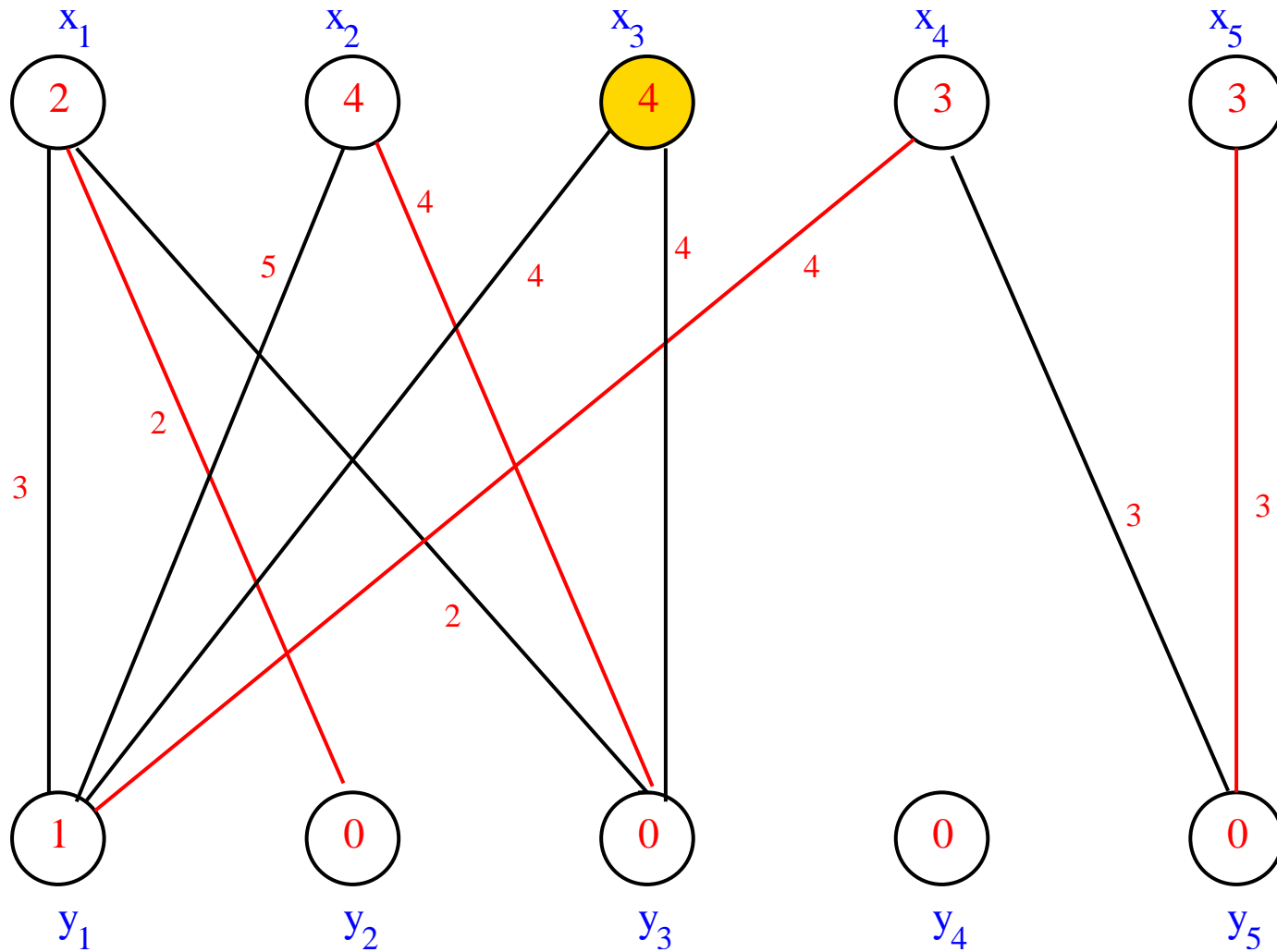
Przykład



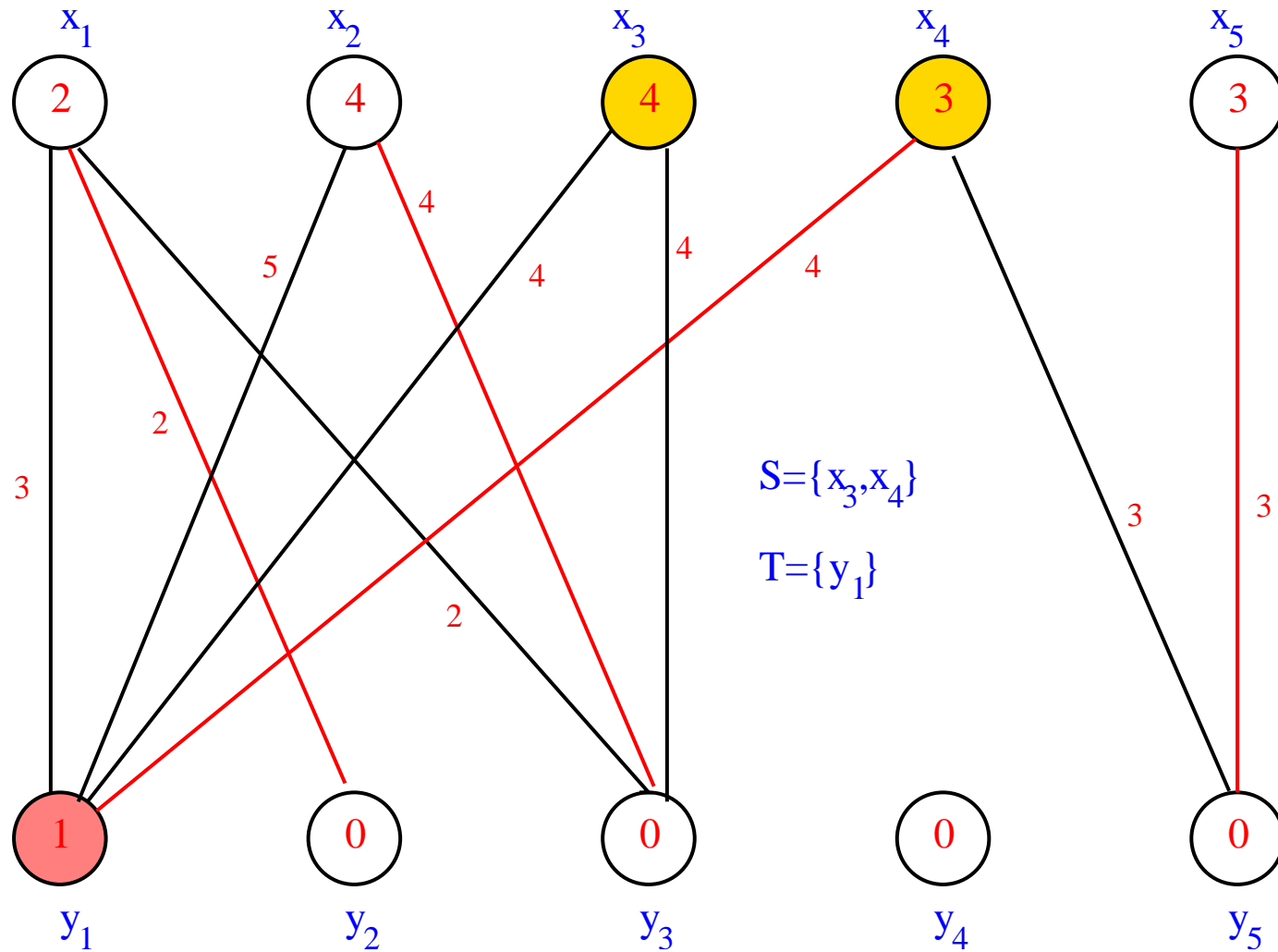
Przykład



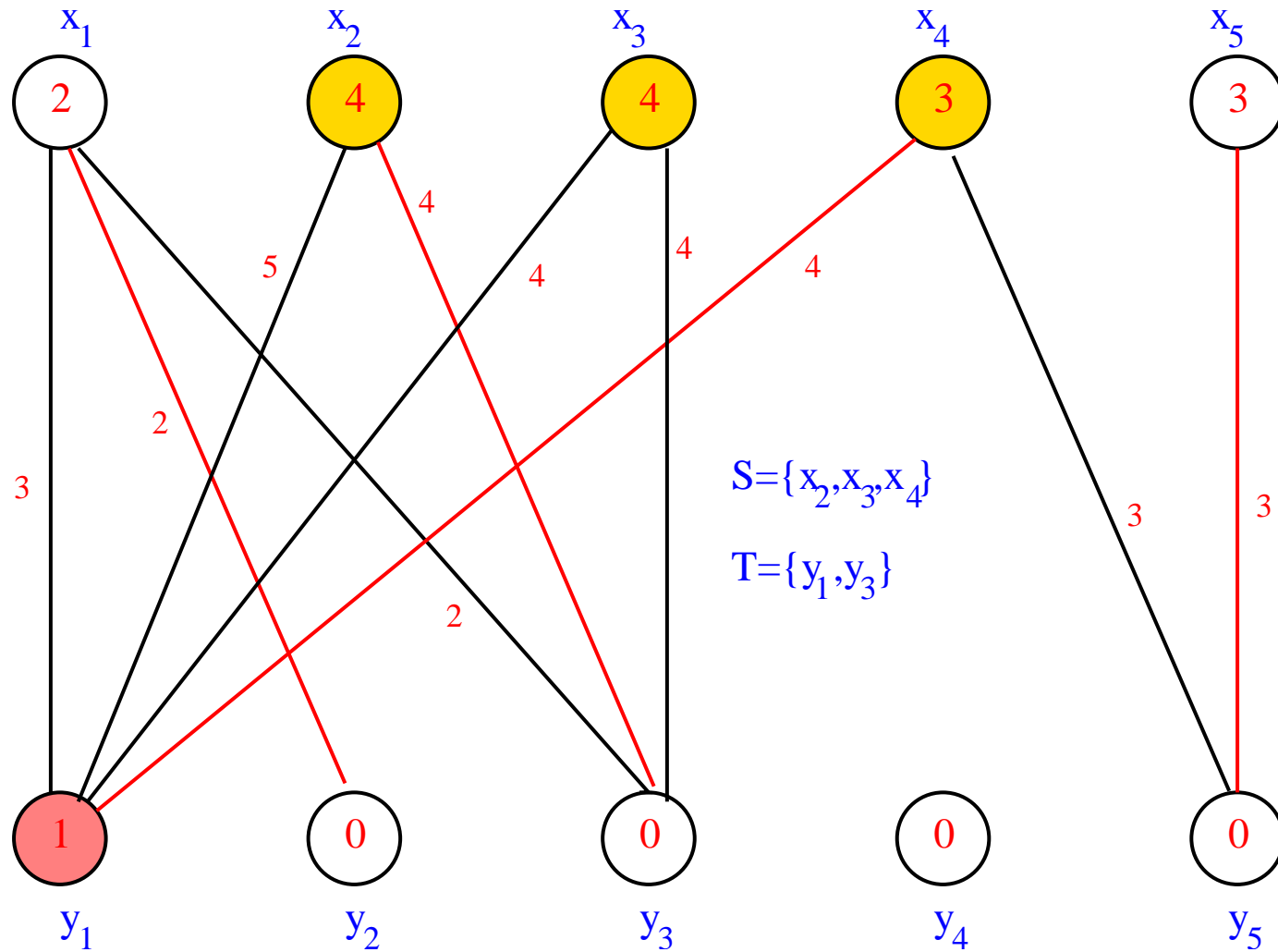
Przykład



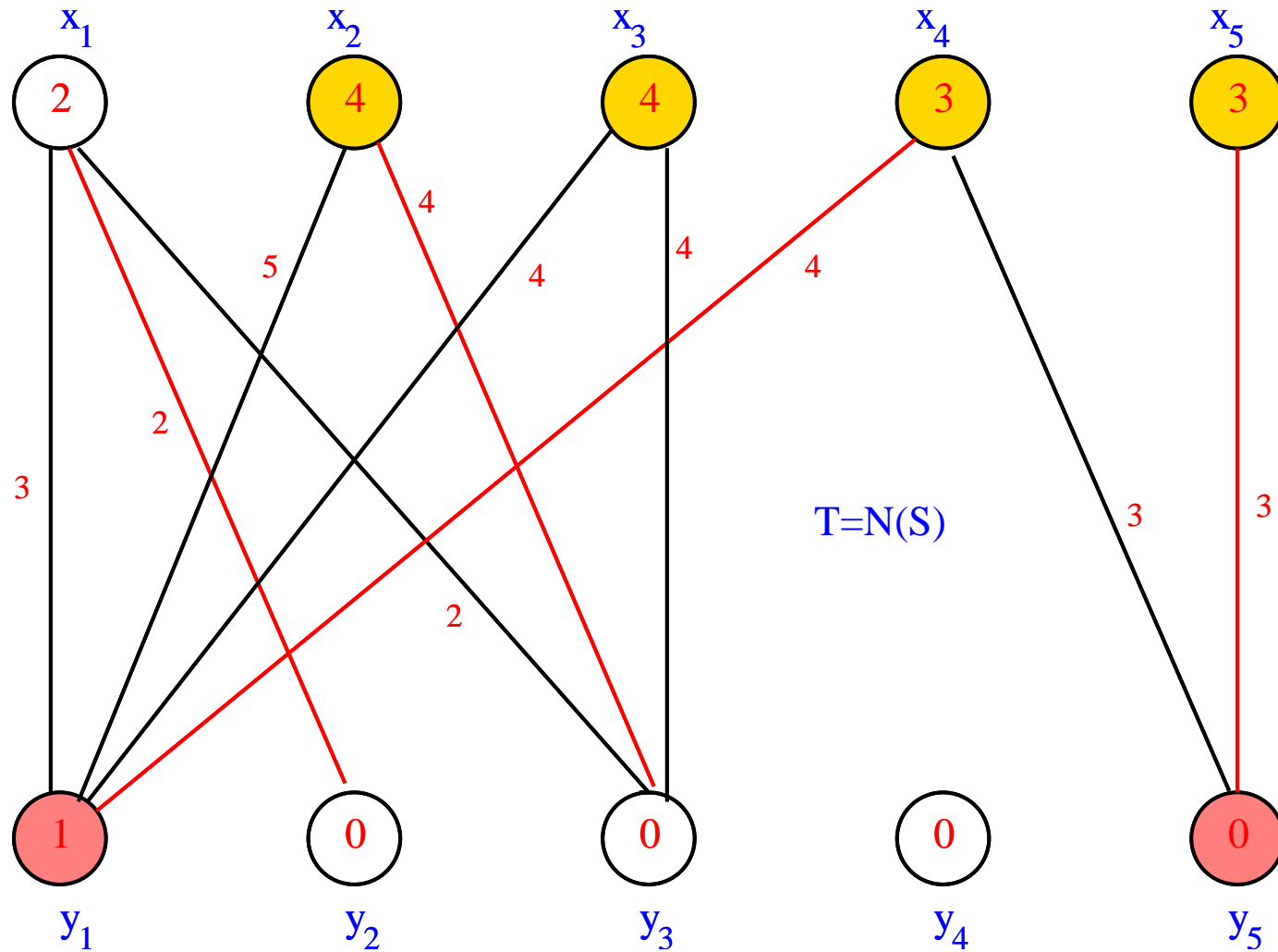
Przykład



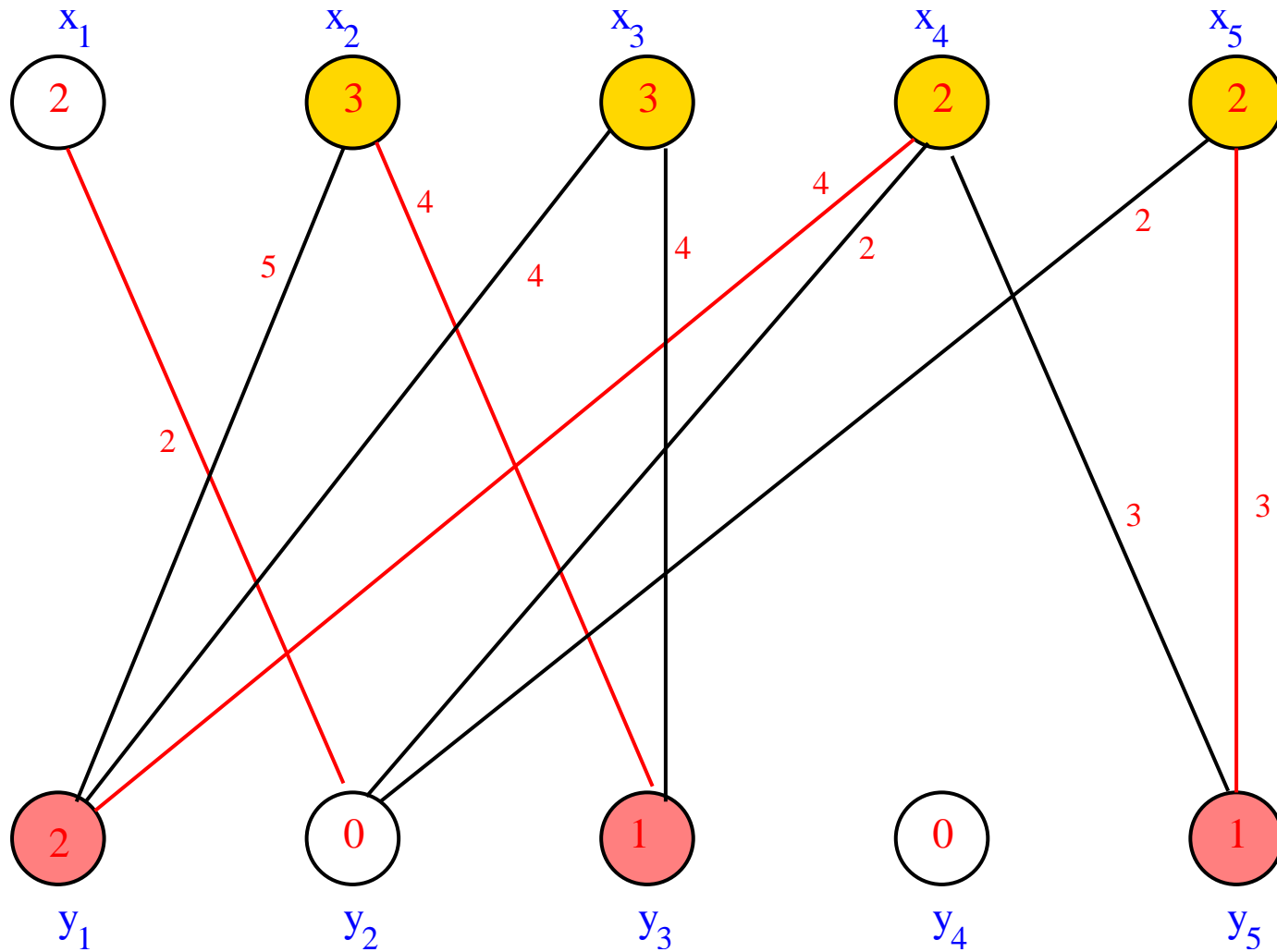
Przykład



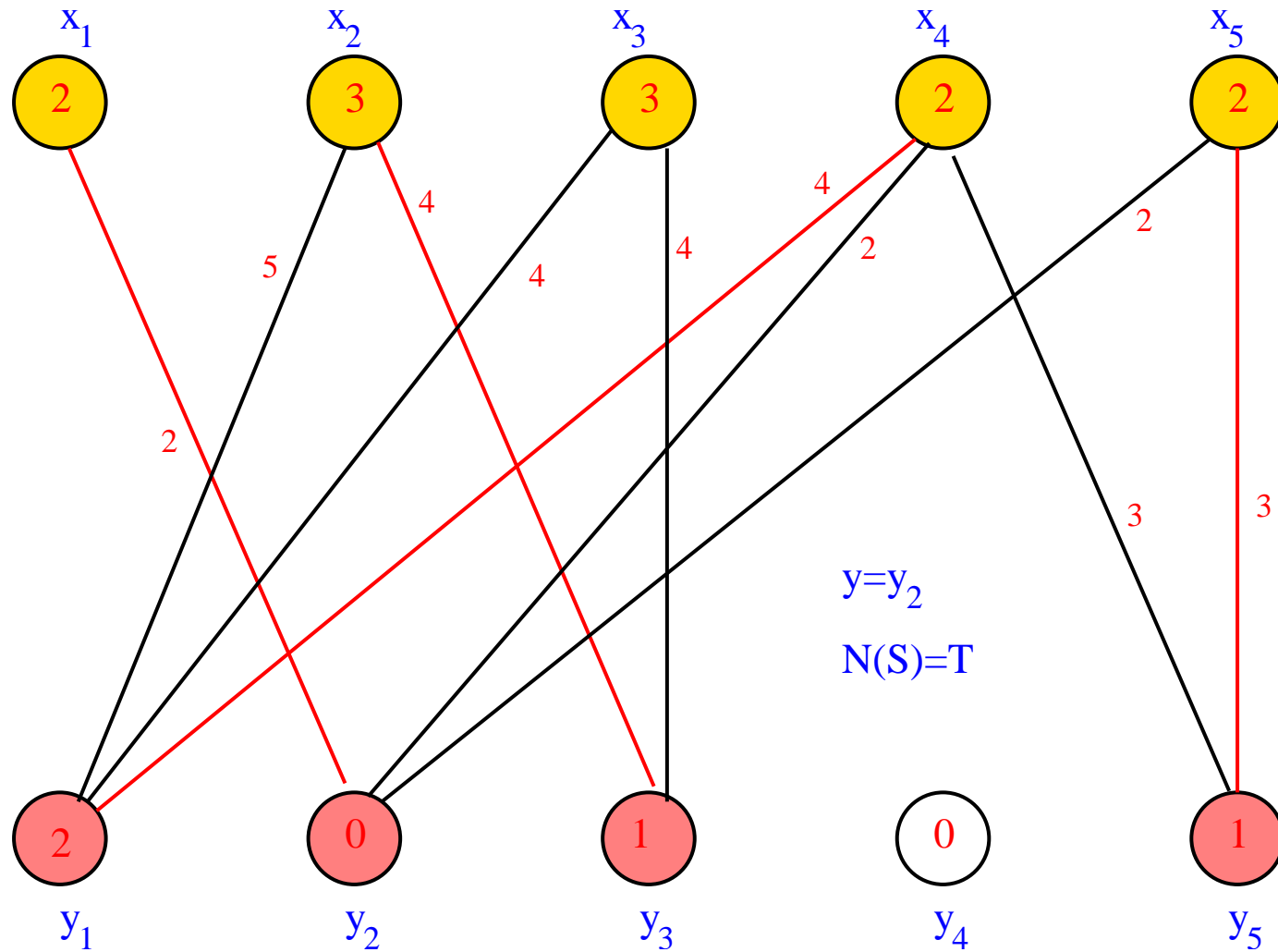
Przykład



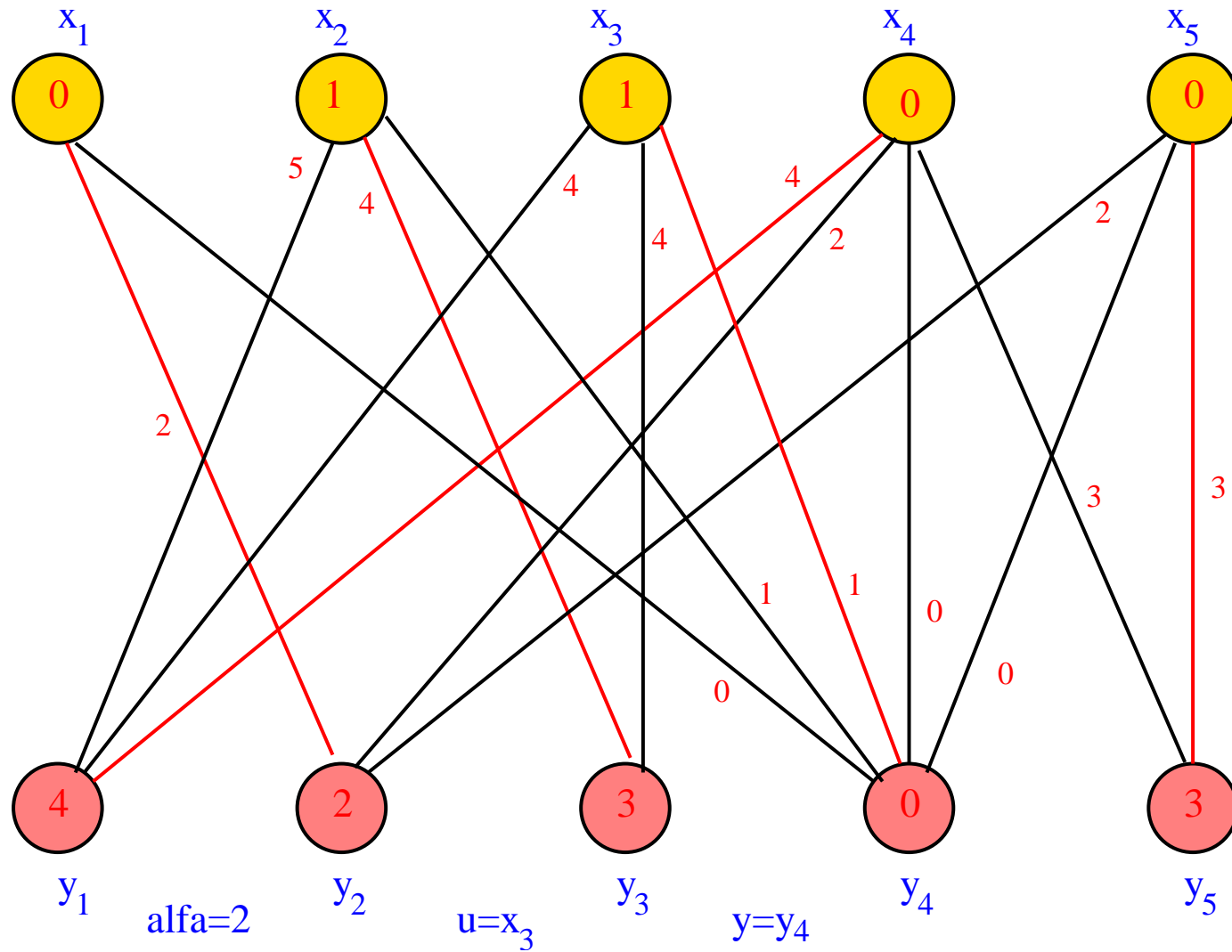
Przykład



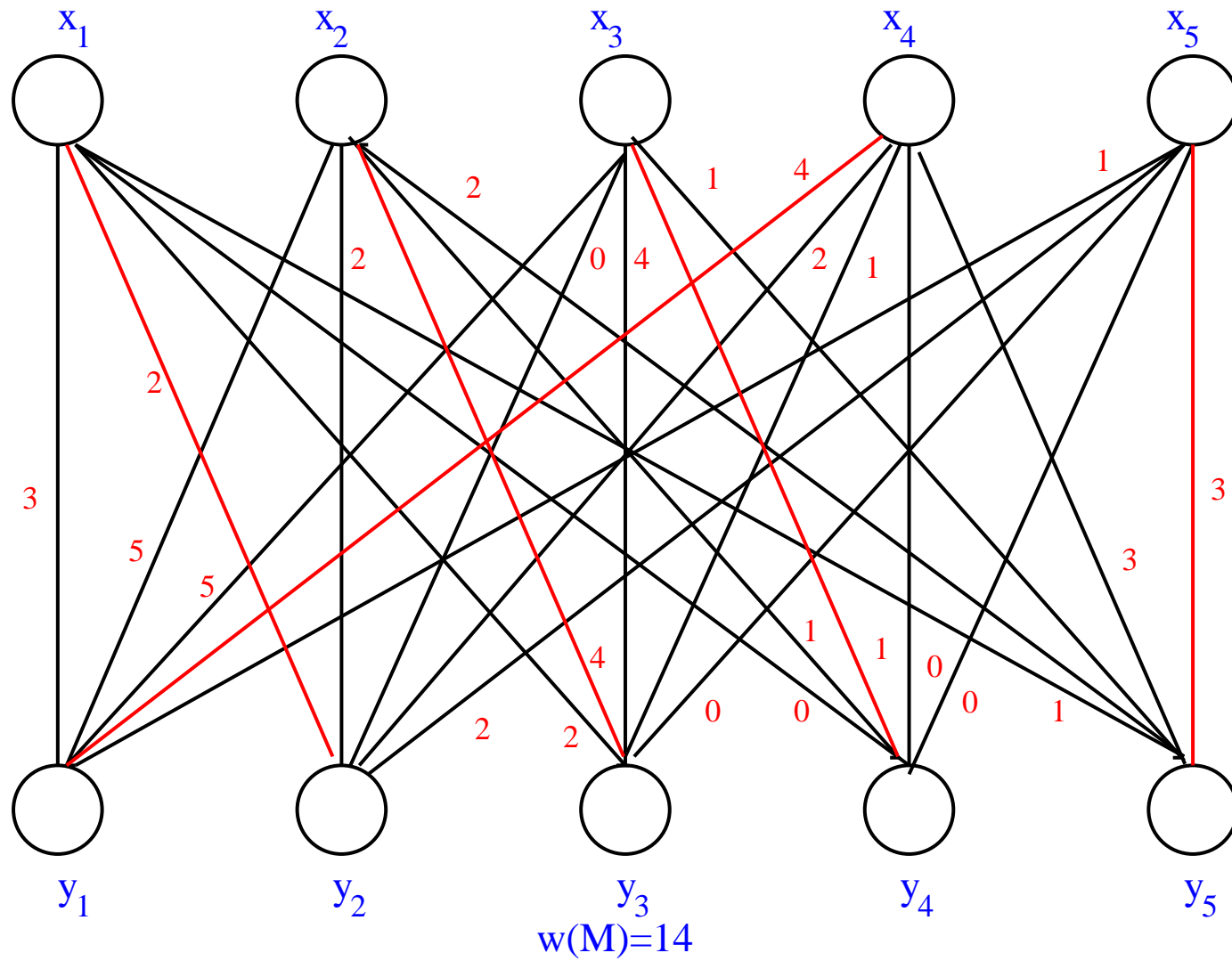
Przykład



Przykład



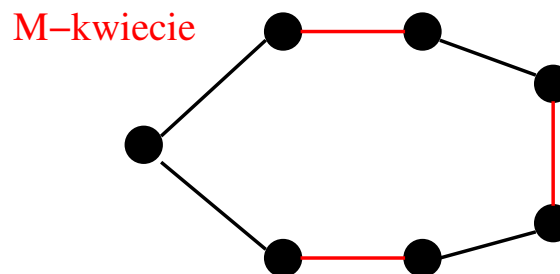
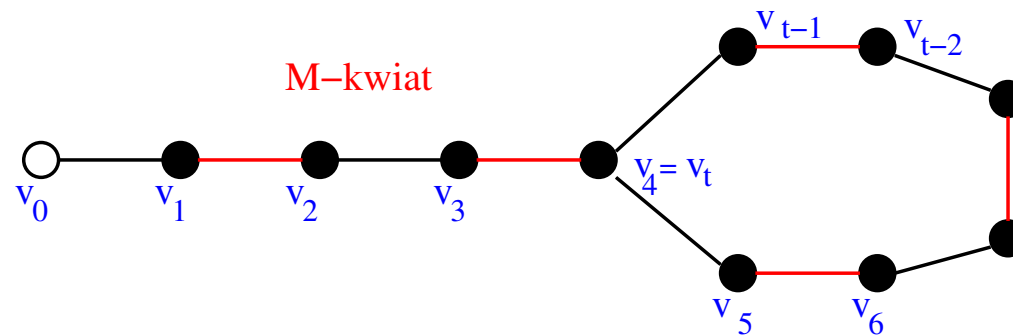
Przykład



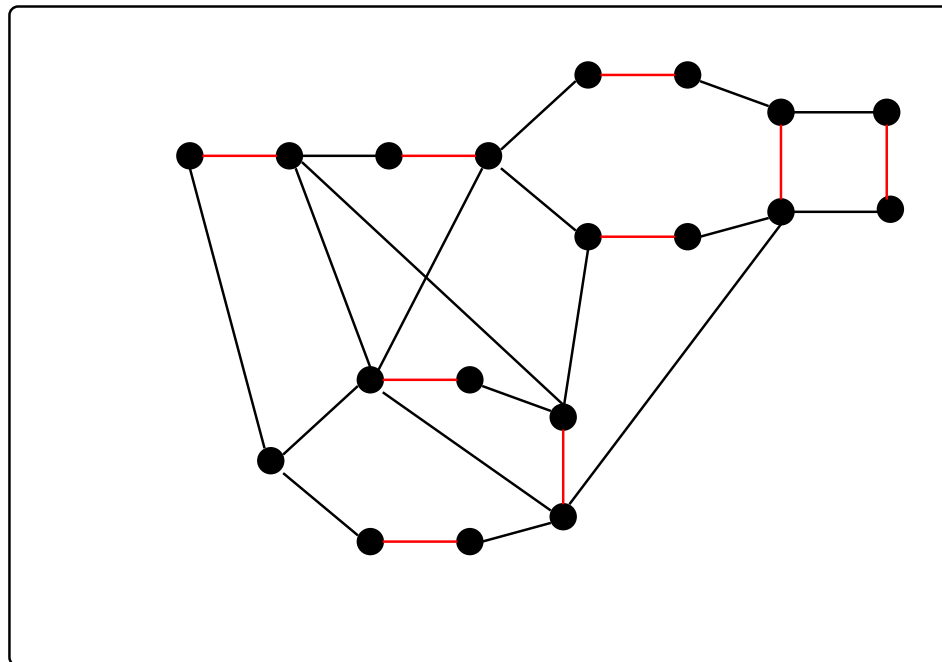
7.4 Skojarzenia w dowolnych grafach

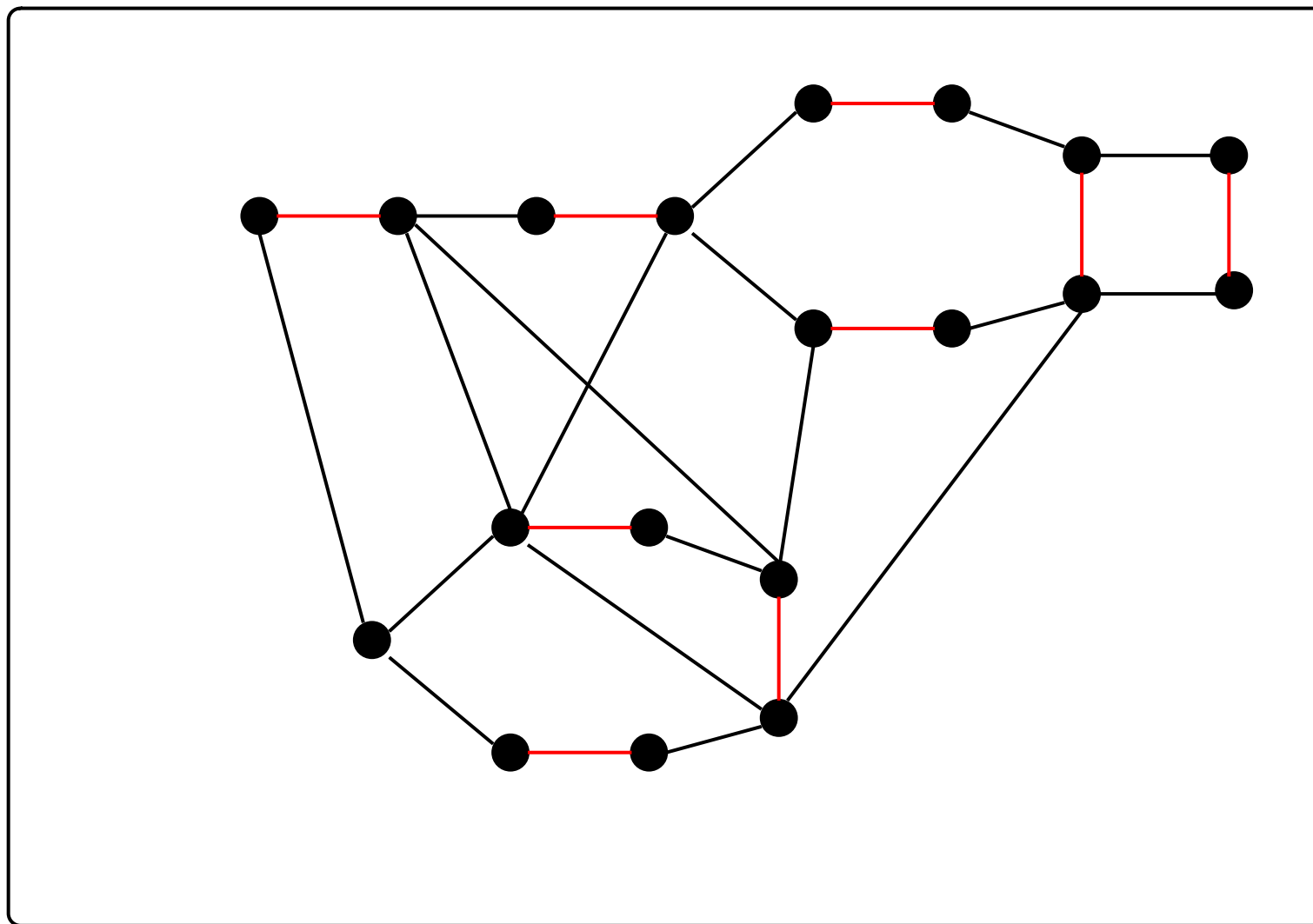
- podstawowa idea stosowana w algorytmach znajdowania skojarzeń o największym rozmiarze polega na znajdowaniu ścieżek M -powiększających
- w przypadku grafów dwudzielnych stosowaliśmy w celu znalezienia takich ścieżek prosty pomysł polegający na odpowiedniej orientacji krawędzi grafu
- w przypadku dowolnych grafów to podejście nie działa ponieważ M -przemienne spacery mogą się “zapętlać” (tworzyć cykle)
- *Edmonds(1965)* rozwiązał ten problem przez “ściąganie” takich cykli do pojedynczego wierzchołka a następnie szukanie największego skojarzenia w mniejszym grafie

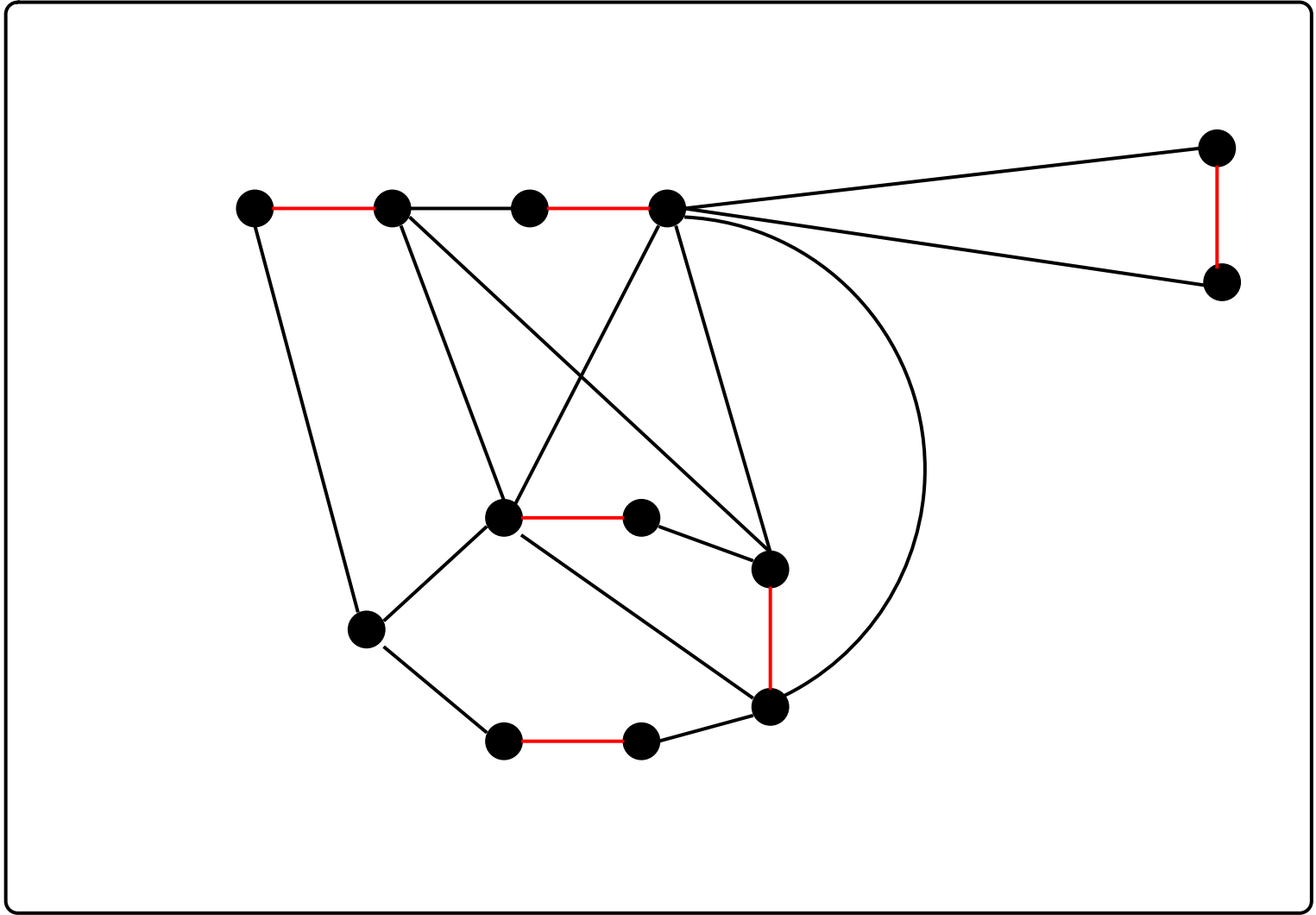
Definicja . M -przemienny spacer $P = (v_0, v_1, \dots, v_t)$ nazywamy M -kwiatem jeżeli t jest nieparzyste, v_0, v_1, \dots, v_{t-1} są wszystkie różne, v_0 jest wierzchołkiem M -nienasyconym, oraz $v_t = v_i$ dla pewnego parzystego $i < t$. Wtedy cykl $(v_i, v_{i+1}, \dots, v_t)$ jest nazywany M -kwieciem.

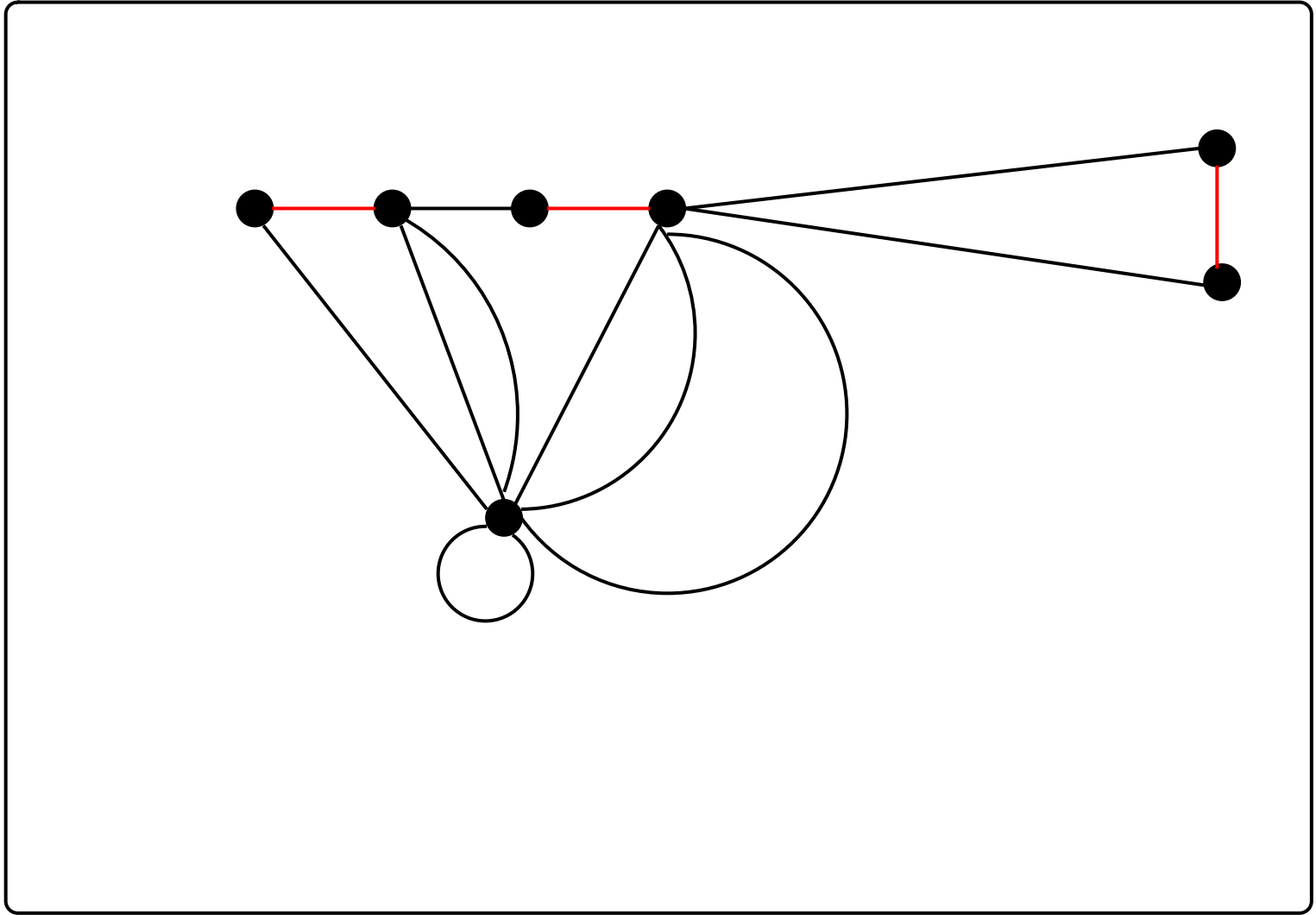


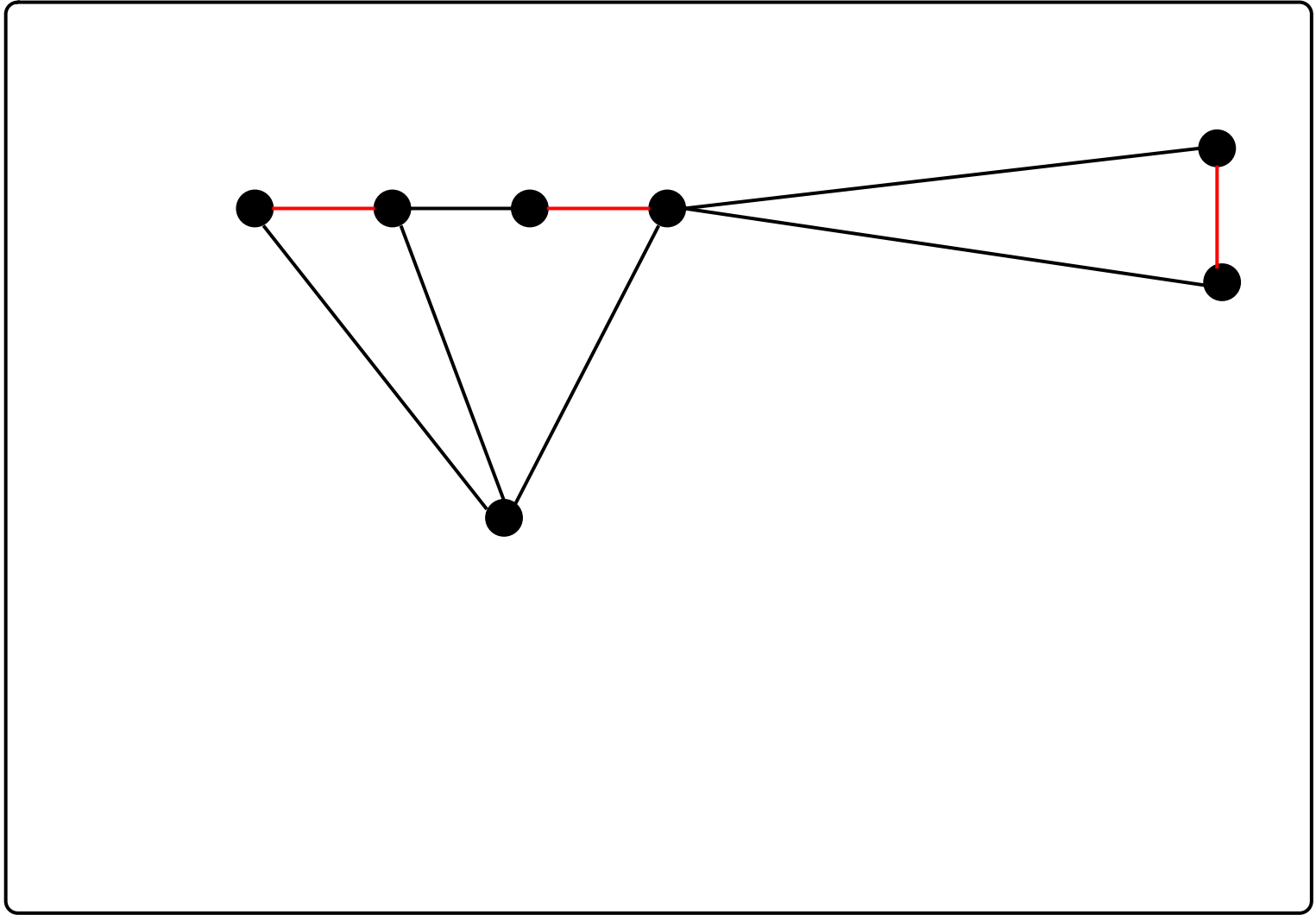
Twierdzenie (Lemat o “ściągnięciu” cykli). Niech G będzie grafem, M jego skojarzeniem, a B będzie M -kwieciem (to znaczy cyklem długości $2k + 1$, który zawiera dokładnie k krawędzi z M). Niech G^* będzie grafem otrzymanym z G poprzez ściągnięcie cyklu B do jednego wierzchołka. Wówczas M jest największym skojarzeniem grafu G wtedy i tylko wtedy, gdy $M^* = M - E(B)$ jest największym skojarzeniem grafu G^* .











Dowód. Załóżmy, że $M^* = M - E(B)$ jest największym skojarzeniem w grafie G^* , natomiast M nie jest największym skojarzeniem w grafie G .

- Z twierdzenia Berge'a wynika, że w grafie G istnieje M -powiększająca ścieżka P .
- Jeżeli P jest rozłączna z cyklem B , to P jest M^* -powiększającą ścieżką w grafie G^* co przeczy założeniu, że M^* jest największe w G^* .
- Jeżeli natomiast P przecina cykl B , to przynajmniej jeden z jej końców nie może należeć do B .
- Powiedzmy, że x jest takim wierzchołkiem końcowym. Niech z będzie pierwszym wierzchołkiem ścieżki P należącym do cyklu B , do którego dochodzimy po tej ścieżce „idąc” od wierzchołka x . Wówczas (x, z) -segment ścieżki P generuje M^* -powiększającą ścieżkę w grafie G^* i podobnie, jak poprzednio, M^* nie mogłoby być największym skojarzeniem.

Dowód. (c.d.) Załóżmy, dowodząc prawdziwość implikacji w drugą stronę, że M^* nie jest największym skojarzeniem grafu G^* i niech w takim razie N^* będzie skojarzeniem w G^* takim, że

$$|N^*| > |M^*|.$$

Zrekonstruujmy cykl B wracając do grafu G . Wówczas N^* będzie generować skojarzenie w grafie G , które nasycza co najwyżej jeden wierzchołek cyklu Z . Możemy więc, używając k krawędzi tego cyklu powiększyć nasze skojarzenie do skojarzenia N o mocy $|N^*| + k$. Stąd

$$|N| = |N^*| + k > |M^*| + k = |M|,$$

czyli M nie mogłoby być wtedy skojarzeniem największym. ■

Twierdzenie . Niech X będzie zbiorem wierzchołków M -nienasyconych, natomiast $P = (v_0, v_1, \dots, v_t)$ będzie najkrótszym M -przemiennym spacerem ze zbioru X do zbioru X . Wtedy albo P jest ścieżką M -powiększającą lub (v_0, v_1, \dots, v_j) jest M -kwiatem dla pewnego $j \leq t$.

Dowód. Przyjmijmy, że P nie jest ścieżką. Wybierzmy możliwie najmniejsze j takie, że $i < j$ oraz $v_j = v_i$. Oznacza to, że $(v_0, v_1, \dots, v_{j-1})$ są wszystkie różne. Jeżeli $j - i$ byłoby parzyste to wtedy można usunąć z P wierzchołki (v_{i+1}, \dots, v_j) , otrzymując w ten sposób krótszy M -przemienny spacer z X do X . Zatem $j - i$ jest nieparzyste. Jeżeli j jest parzyste a i jest nieparzyste to $v_{i+1} = v_{j-1}$ (ponieważ jest wierzchołkiem skojarzonym z $v_i = v_j$, co przeczy minimalności wyboru j). Zatem j jest nieparzyste a i jest parzyste, co implikuje iż (v_0, v_1, \dots, v_j) jest M -kwiatem. ■

Algorytm powiększania skojarzenia

Dane: graf $G = (V, E)$ oraz skojarzenie M

Poszukiwane: M -powiększająca ścieżka (o ile istnieje)

Oznaczmy przez X zbiór wierzchołków M -nienasyconych w G .

1. Jeżeli nie istnieje w G spacer M -przemienny z X do X to G nie zawiera M -powiększającej ścieżki – STOP
2. Jeżeli istnieje w G spacer M -przemienny z X do X dodatniej długości to wybierz najkrótszy możliwy, powiedzmy, $P = (v_0, v_1, \dots, v_t)$.

- a Jeżeli P jest ścieżką to podaj na wyjście P – STOP
- b Jeżeli P nie jest ścieżką to wybierz j takie, że (v_0, v_1, \dots, v_j) jest M -kwiatem z M -kwieciami B .
Zastosuj algorytm (rekurencyjnie) do grafu $G \circ B$ (graf powstały z G po “ściągnięciu” B) i skojarzenia $M \circ B$, otrzymując w ten sposób $M \circ B$ -powiększającą ścieżkę P w $G \circ B$. Następnie rozszerz P do M -powiększającej ścieżki w G – STOP

Twierdzenie . Dla dowolnego grafu G istnieje algorytm znajdujący skojarzenie o największym rozmiarze w czasie $O(n^2m)$.

Dowód.

- Algorytm znajdowania skojarzenia o największej mocy jest prostą konsekwencją powyższego algorytmu. Rozpoczynając z $M = \emptyset$ można, iterując, znaleźć M -powiększającą ścieżkę P a następnie zastąpić M przez $M \div E(P)$. Algorytm kończy pracę jeżeli w grafie G nie ma już żadnej ścieżki M -powiększającej.
- Ścieżkę P można znaleźć w czasie $O(m)$ a graf $G \circ B$ w czasie $O(m)$. Ponieważ rekurencja ma głębokość co najwyżej n , dlatego M -powiększającą ścieżkę P można znaleźć w czasie $O(nm)$. Końcowa złożoność wynika z faktu, że liczba “ulepszeń” nie może przekroczyć $n/2$.

Algorytm o czasie $O(n^3)$

Poprzedni algorytm, w każdej iteracji, prowadzi do powiększania skojarzenia M . Krok iteracyjny składa się z dwóch czynności:

1. znalezienia M -przemennego spaceru i
2. "ściągnięcia" M -kwiecia,

powtarzanych tak długo aż otrzymany M -spacer jest prosty, to znaczy jest ścieżką M -powiększającą.

Przypomnijmy, że czynności 1 oraz 2 wykonujemy w czasie $O(m)$ (co w rezultacie prowadzi do algorytmu o złożoności $O(n^2m)$).

Dlatego aby "przyspieszyć" algorytm znajdowania największego skojarzenia należy zoptymalizować obie te czynności, stosując "lokalne" modyfikacje grafu.

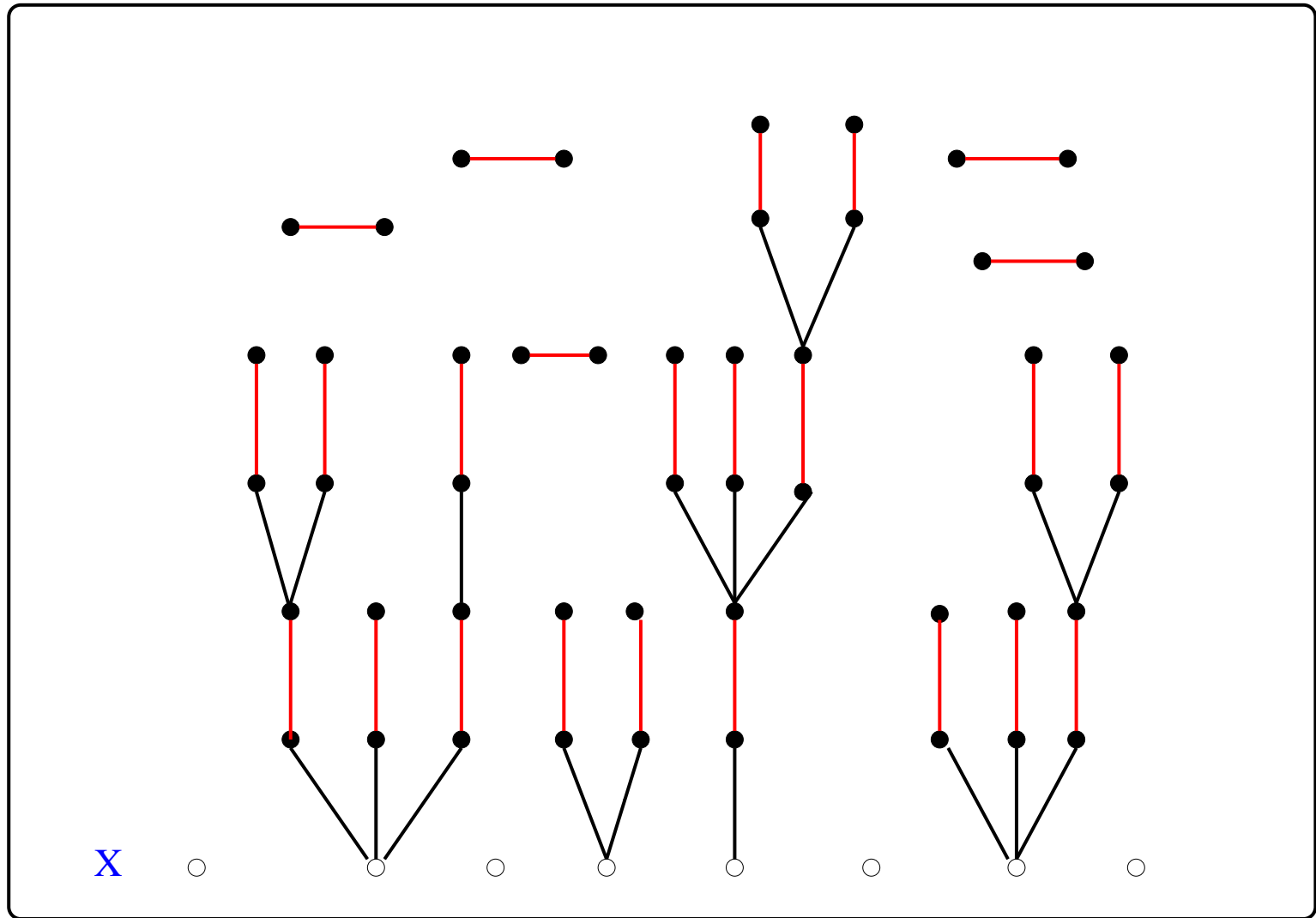
Niech $G = (V, E)$ będzie grafem prostym, a M pewnym skojarzeniem w G . Oznaczmy, jak poprzednio, przez X zbiór wierzchołków M -nienasyconych w G .

Definicja . Podgraf F grafu G nazywamy lasem M -przemiennym, jeżeli F jest lasem takim, że $M \subseteq F$, każda składowa (drzewo) lasu F zawiera dokładnie jeden wierzchołek (korzeń) z X lub tworzy ją jedna krawędź z M , oraz każda ścieżka rozpoczynająca się w X jest M -przemienna.

Definicja .

- $parzyste(F) := \{v \in V : F \text{ zawiera parzystą } X - v \text{ ścieżkę}\}$
- $nieparzyste(F) := \{v \in V : F \text{ zaw. nieparzystą } X - v \text{ ścieżkę}\}$
- $wolne(F) := \{v \in V : F \text{ nie zawiera } X - v \text{ ścieżki}\}$

M -przemienny las



Zauważmy, że

- korzenie drzewa F należą do $parzyste(F)$
- jeżeli $u \in nieparzyste(F)$ to u jest incydentne z dokładnie jedną krawędzią z M i dokładnie jedną krawędzią z $F \setminus M$ (każdy wierzchołek drzewa w odległości nieparzystej od korzenia ma stopień dwa)
- ze wzoru Tutte'a-Berge'a wynika, że jeżeli nie istnieje krawędź łącząca $parzyste(F)$ z $parzyste(F) \cup wolne(F)$, to M jest skojarzeniem o największej mocy w G

Twierdzenie (wzór Tutte'a-Berge'a).

Dla każdego grafu $G = (V, E)$,

$$\alpha'(G) = \min_{U \subseteq V} \frac{1}{2}(|V| + |U| - o(G - U))$$

Dowód. (tylko \leq) Zauważmy, że dla każdego $U \subseteq V$ moc największego skojarzenia w G

$$\begin{aligned} \alpha'(G) &\leq |U| + \alpha'(G - U) \leq |U| + \frac{1}{2}(|V \setminus U| - o(G - U)) \\ &= \frac{1}{2}(|V| + |U| - o(G - U)) \end{aligned}$$

- jeżeli nie istnieje krawędź łącząca $parzyste(F)$ z $parzyste(F) \cup wolne(F)$, to M jest skojarzeniem o największej mocy w G
- jeżeli taka krawędź nie istnieje to $parzyste(F)$ jest zbiorem niezależnym w $G - nieparzyste(F)$
- biorąc $U := nieparzyste(F)$, mamy

$$o(G - U) \geq |parzyste(F)| = |X| + |U| = |V| - 2|M| + |U|$$

- $|M| \geq \frac{1}{2}(|V| + |U| - o(G - U))$
- zatem ze wzoru Tutte'a-Berge'a wynika, że M jest największym skojarzeniem w G .

Struktury danych:

- M -przemienny las
- dla każdego wierzchołka v pamiętamy krawędzie z E , M oraz F z nim incydentne, oraz jedną krawędź $e_v = vu$, gdzie $u \in \text{parzyste}(F)$ (o ile taka krawędź istnieje)
- oraz funkcje indykatorową przynależności do zbiorów $\text{parzyste}(F)$ i $\text{nieparzyste}(F)$

Algorytm znajdowania największego skojarzenia

Dane: graf $G = (V, E)$ oraz skojarzenie M

Poszukiwane: największe skojarzenie M w G

Oznaczmy przez X zbiór wierzchołków M -nienasyconych w G , $F \leftarrow M$, i dla każdego $v \in V$ wybierz krawędź $e_v = vu$, gdzie $u \in \text{parzyste}(F)$, gdzie $u \in \text{parzyste}(F)$.

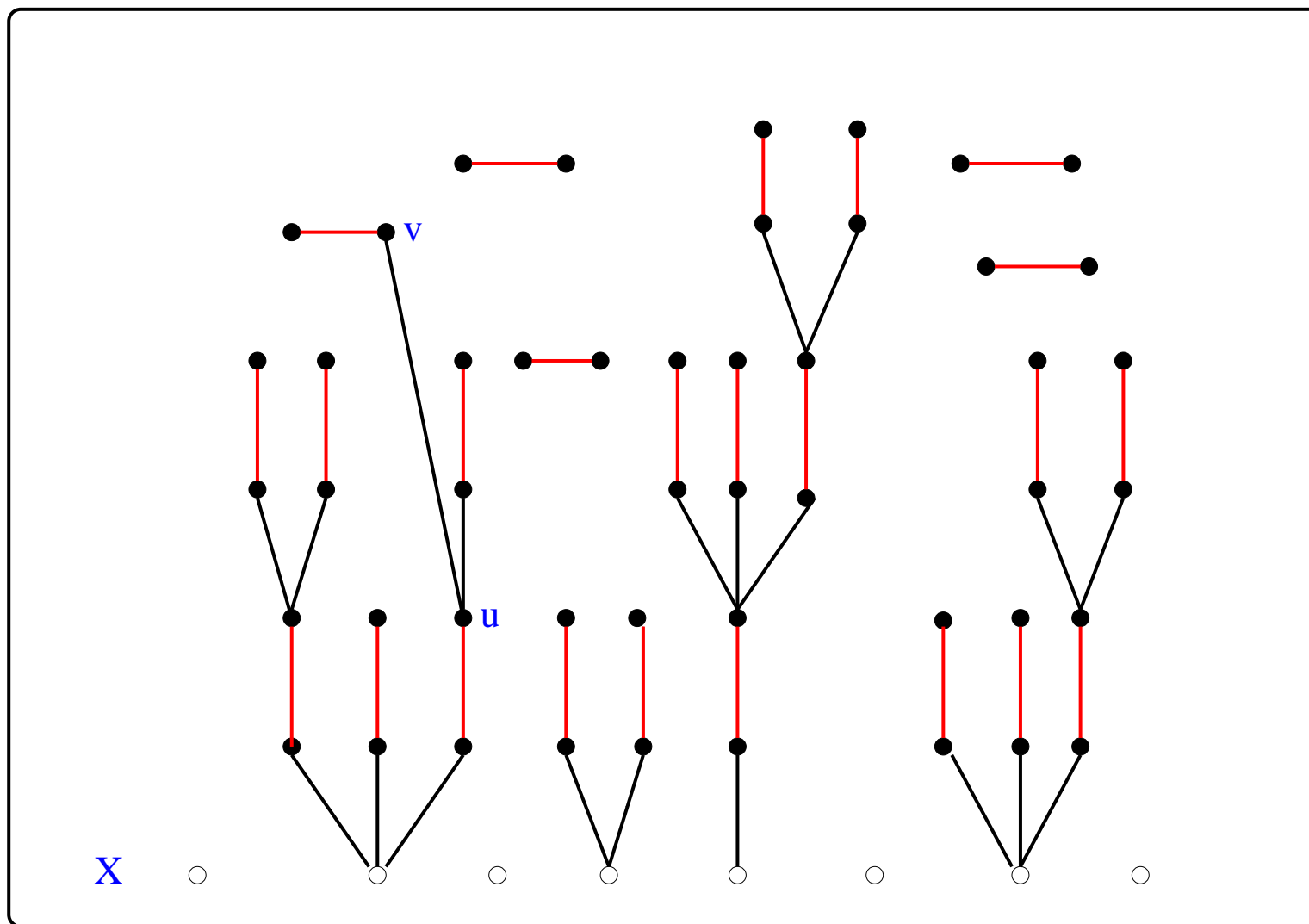
Znajdź wierzchołek $v \in \text{parzyste}(F) \cup \text{wolne}(F)$ dla którego istnieje krawędź $e_v = vu$.

1. $v \in \text{wolne}(F)$. Dodaj krawędź uv do F . Niech vw będzie krawędzią z M incydentną z v . Dla każdej krawędzi wx incydentnej z w wykonaj $e_x \leftarrow wx$.

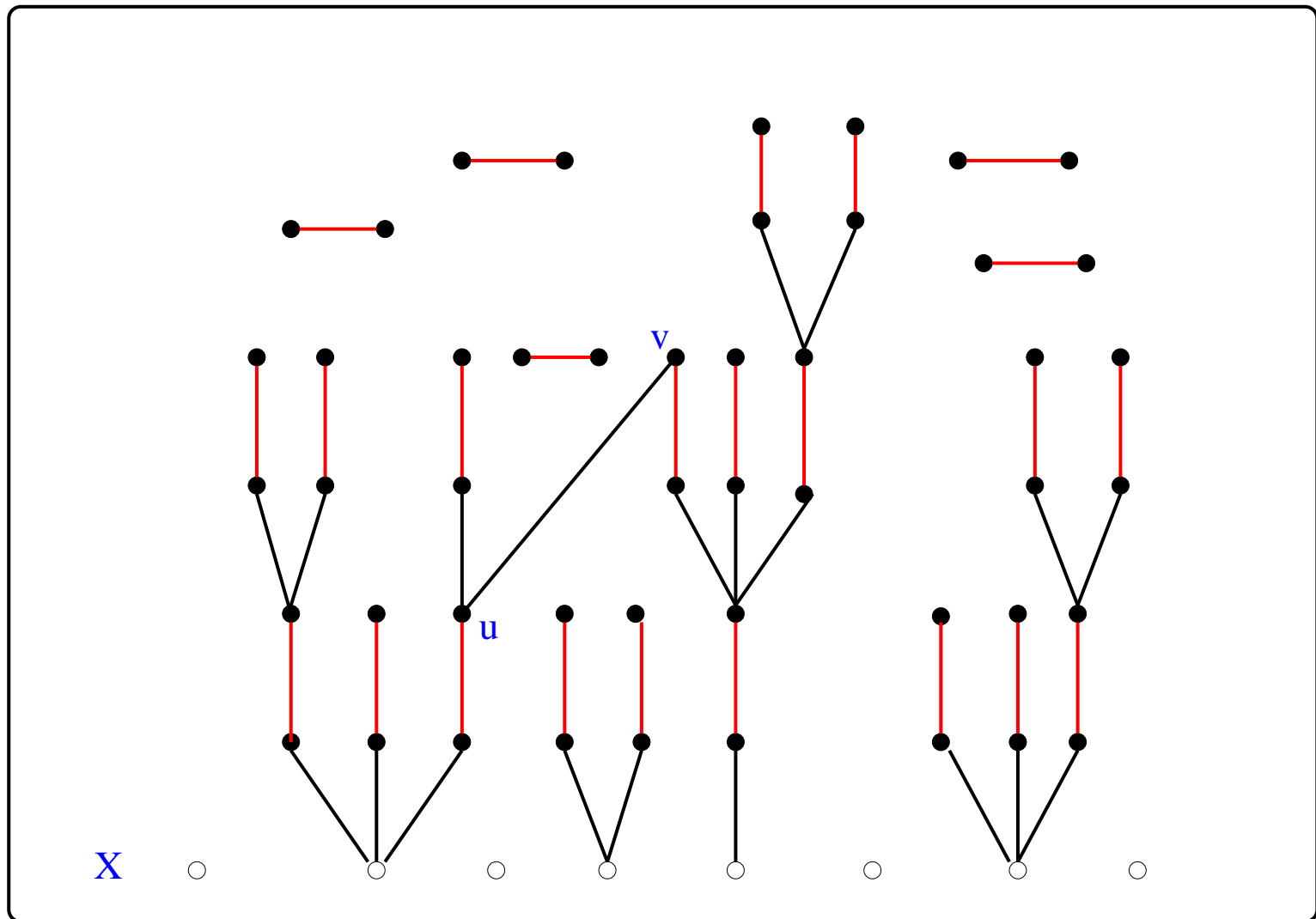
2. $v \in \text{parzyste}(F)$. Znajdź $X - u$ oraz $X - v$ ścieżki P i Q w F .

- (a) Ścieżki P i Q są rozłączne. Wtedy P i Q razem z krawędzią uv tworzą ścieżkę M -powiększającą.
- (b) Ścieżki P i Q nie są rozłączne. Wtedy P i Q zawierają M -kwiecie B . Dla każdej krawędzi bx , gdzie $b \in B$ oraz $x \notin B$ podstaw $e_x \leftarrow Bx$. Zastąp G przez $G \circ B$ oraz usuń wszystkie pętle i krawędzie równoległe z E, M oraz F .

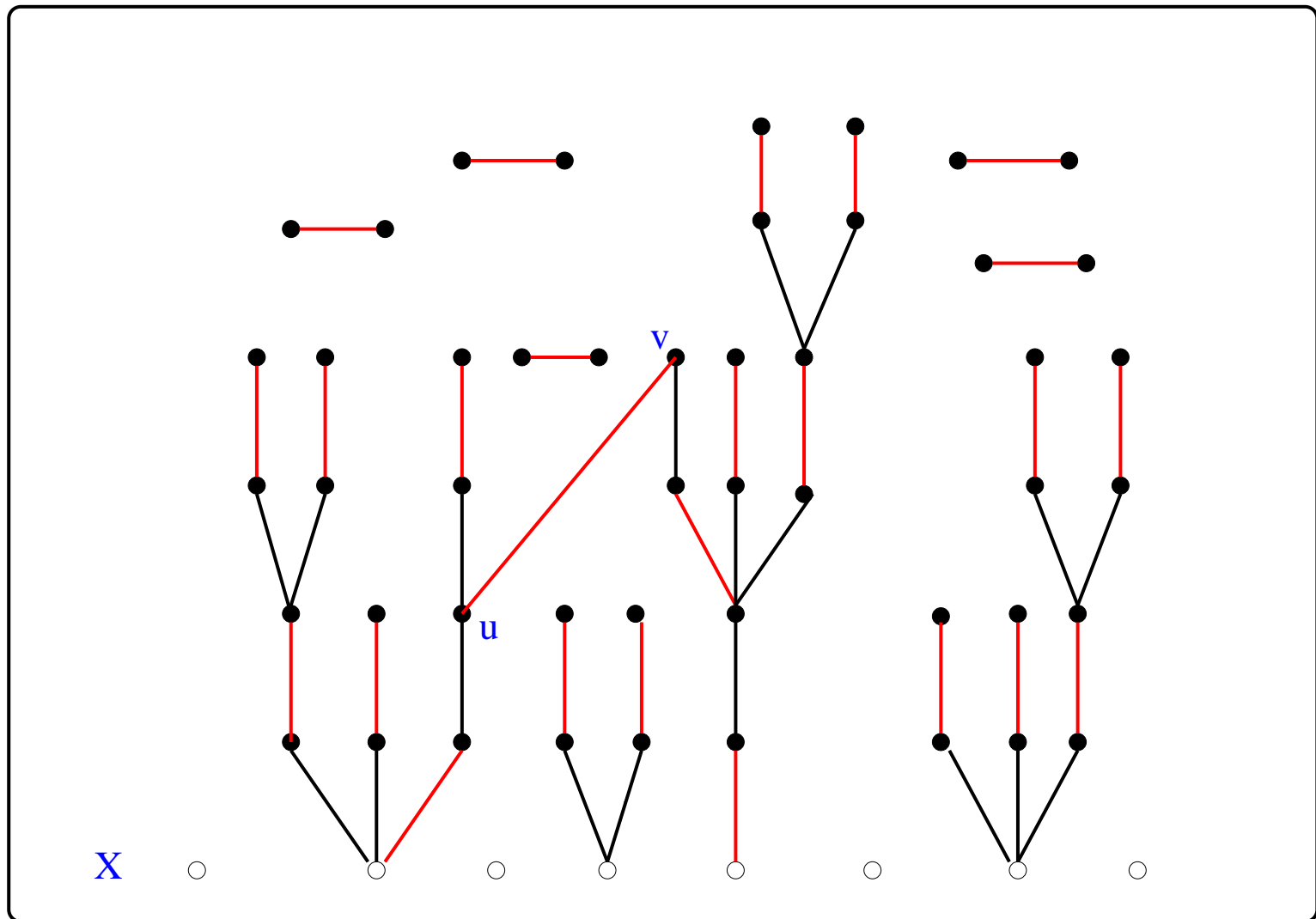
Przypadek 1



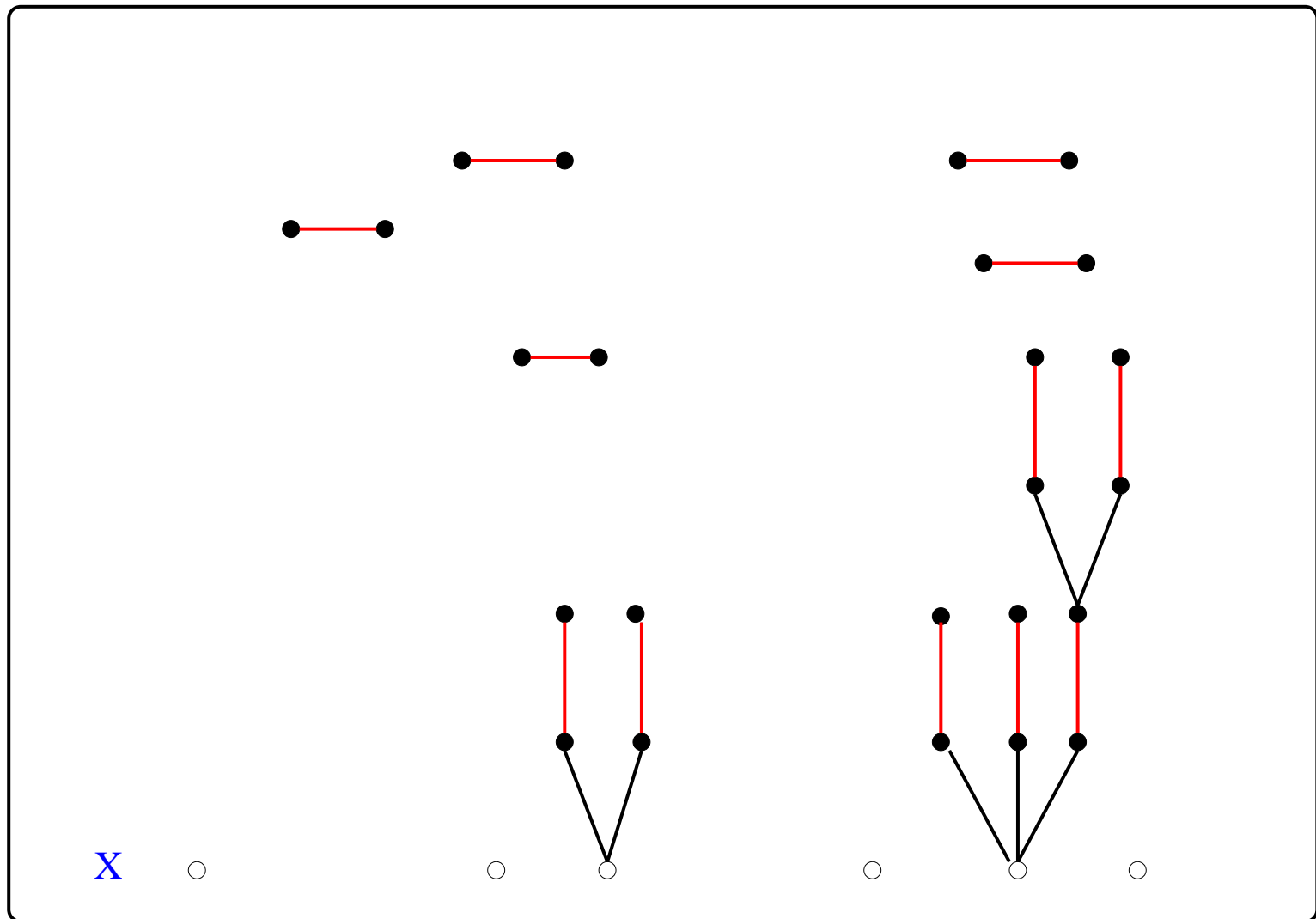
Przypadek 2a



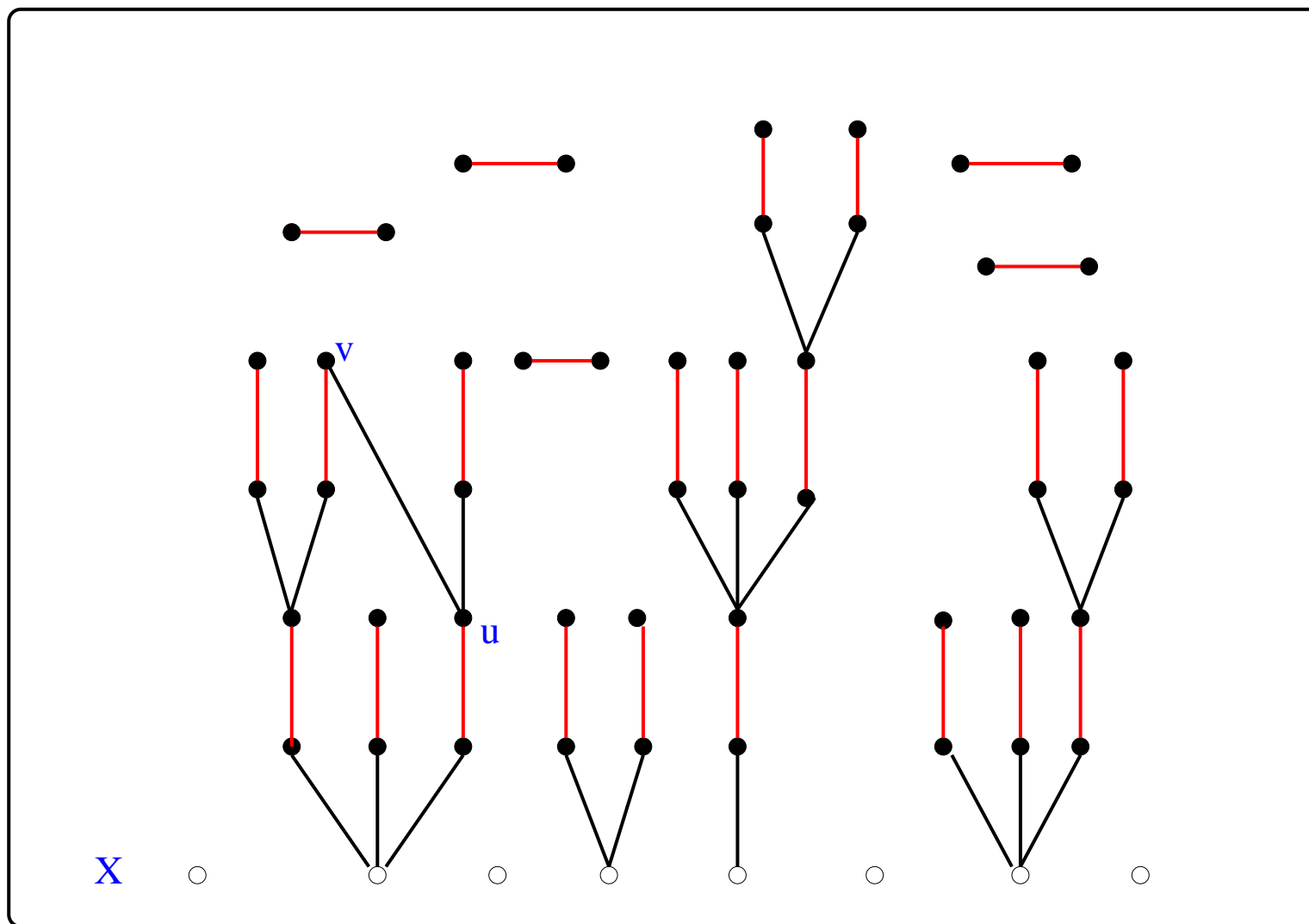
Przypadek 2a



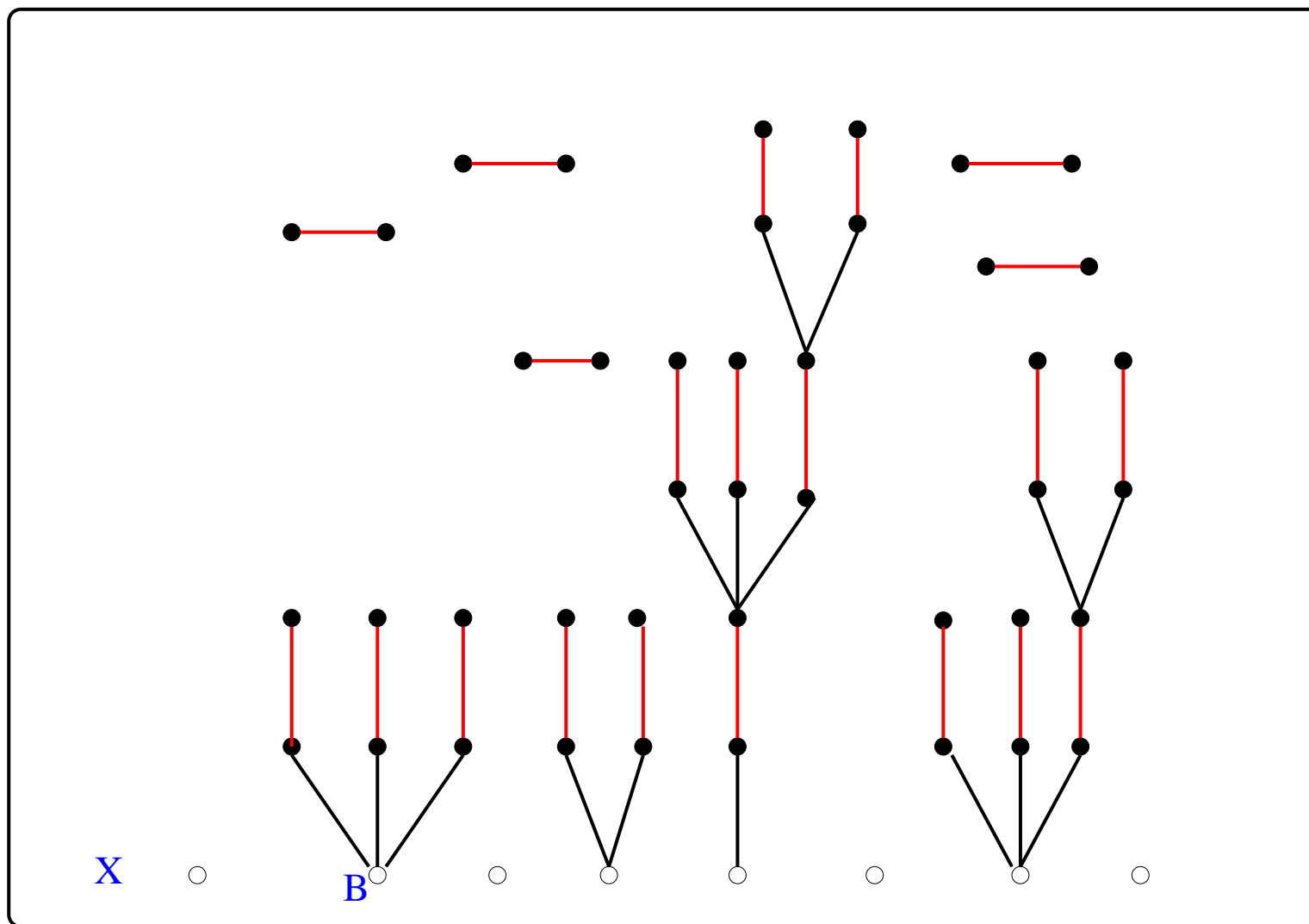
Przypadek 2a



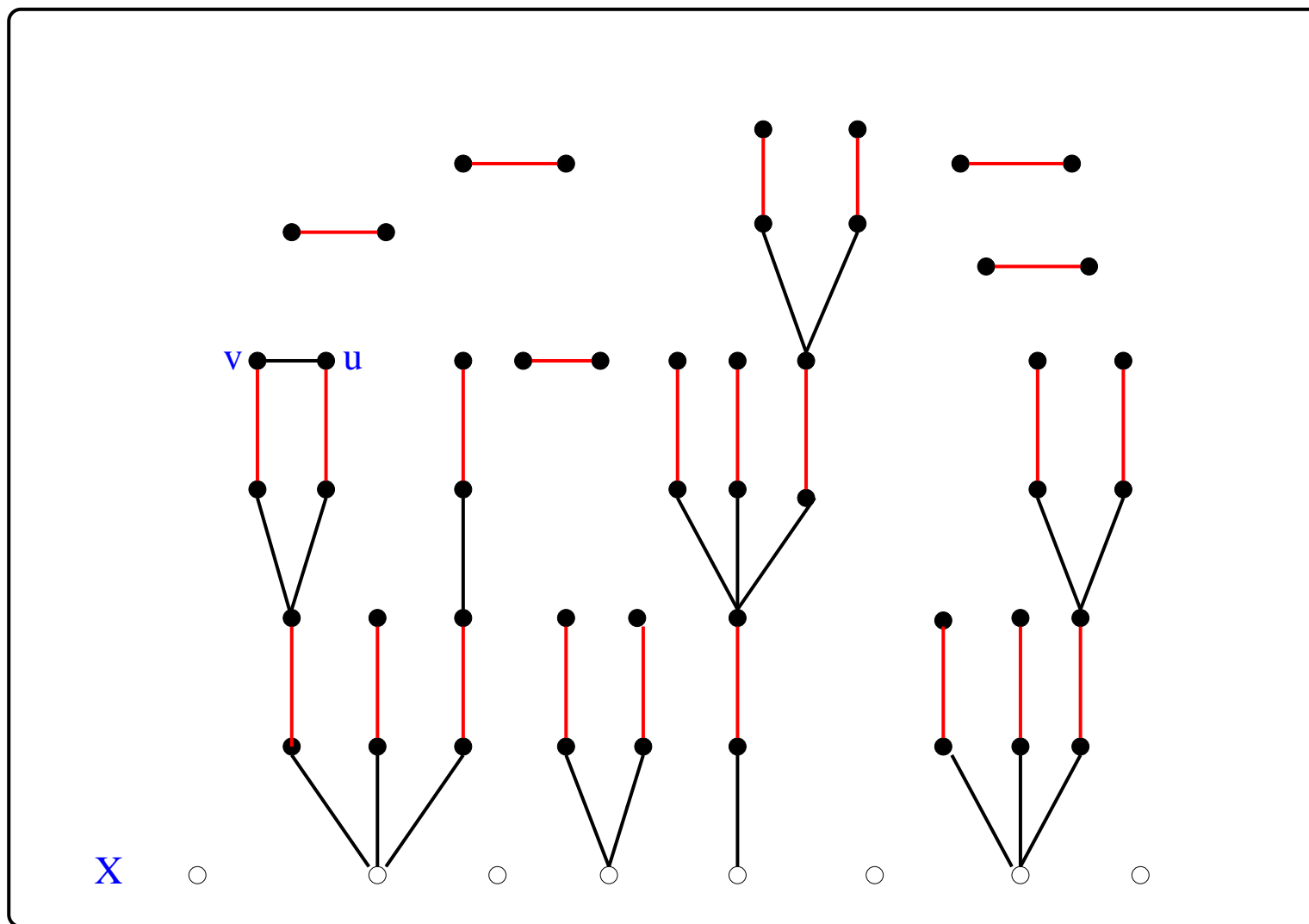
Przypadek 2b-1



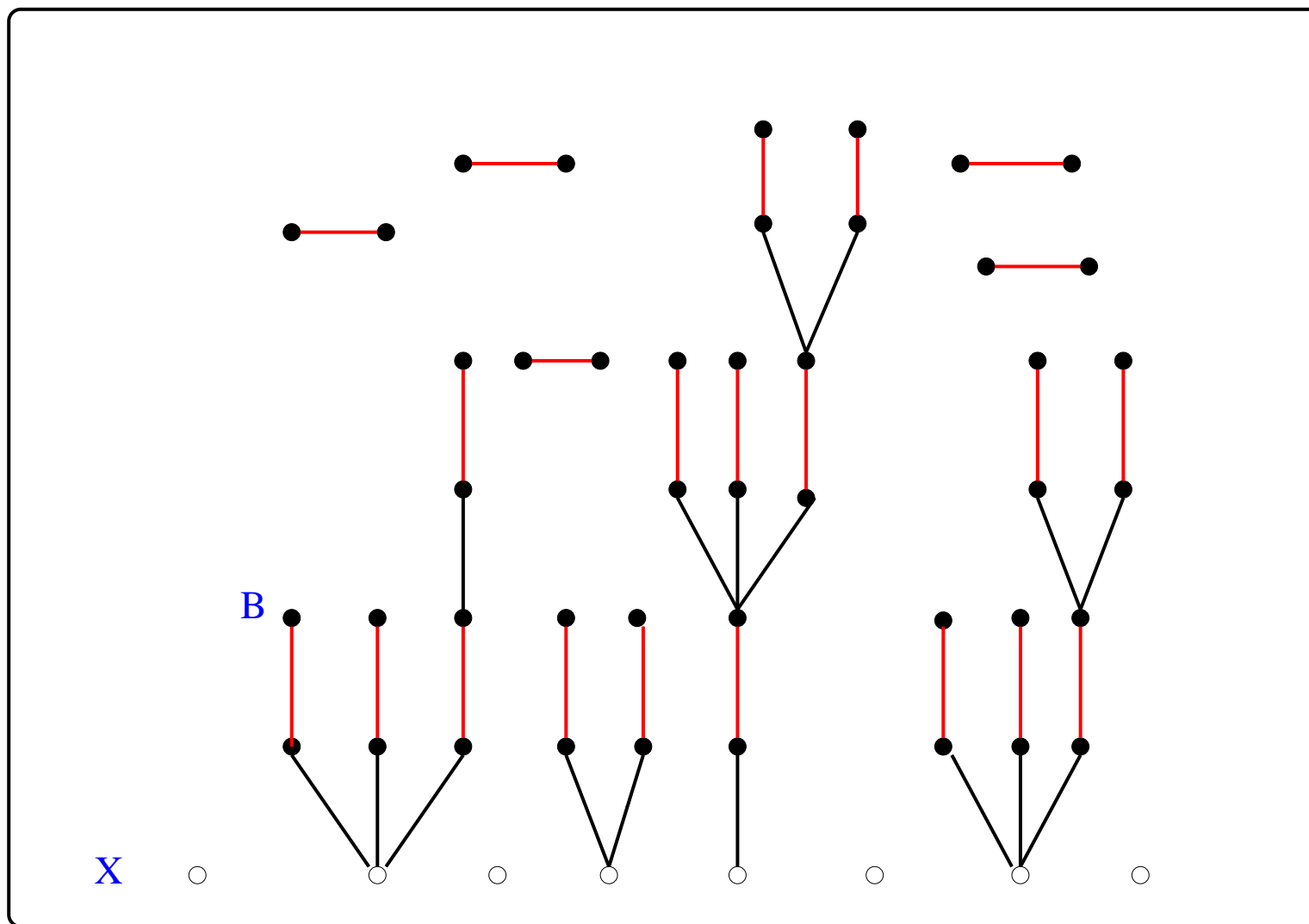
Przypadek 2b-1



Przypadek 2b-2



Przypadek 2b-2




Twierdzenie . Algorytm znajduje największe skojarzenie w dowolnym grafie G w czasie $O(n^3)$.

Dowód. Liczba iteracji głównej pętli algorytmu nie przekracza $|V|$, ponieważ w każdej iteracji, $|V| + \text{wolne}(F)$ zmniejsza się co najmniej o dwa (jeden ze składników maleje o co najmniej dwa podczas gdy drugi pozostaje bez zmian).

Przypadek 1. Uaktualnienie struktury danych w tym przypadku zajmuje $O(n)$.

Przypadek 2a. Ścieżki P i Q można znaleźć w czasie $O(n)$, zatem całkowity czas wyznaczenia ścieżki M -powiększającej jest równe $O(n)$.

Przypadek 2b. Uaktualnienie struktury danych w tym przypadku zajmuje $O(|B|n)$. Podobnie zamiana G na $G \circ B$ zajmuje $O(|B|n)$. Ponieważ $|B|$ jest ograniczony z góry przez podwojone obniżenie liczby wierzchołków grafu, dlatego całkowity czas wynosi $O(n^2)$. 

7.5 Skojarzenia w dowolnych ważonych grafach

- podobnie jak w przypadku dowolnych grafów bez wag na krawędziach algorytm dla grafów z wagami (grafów ważonych) wykorzystuje ideę “ściągnięcia” nieparzystych cykli (M -kwiecia) oraz powiększania skojarzeń za pomocą ścieżek M -przemiennych
- w przypadku grafów ważonych będziemy również stosować operację przeciwną do “ściągnięcia” czyli “rozciąganie” cykli
- podamy algorytm znajdowania skojarzenia **doskonałego** o **minimalnej** wadze w grafie (który implikuje algorytm znajdowania skojarzenia o maksymalnej wadze)

- Niech $G = (V, E)$ będzie grafem a $w : E \rightarrow \mathbb{Q}$, gdzie \mathbb{Q} oznacza zbiór liczb wymiernych, będzie funkcją ważenia krawędzi grafu G .
- Zakładamy, że G ma co najmniej jedno skojarzenie doskonałe oraz, że wagi są nieujemne

Definicja . Rodzinę podzbiorów \mathcal{C} zbioru V nazywamy *warstwową* jeżeli dla każdej pary zbiorów $T, U \in \mathcal{C}$, $T \subseteq U$ lub $U \subseteq T$ lub $T \cap U = \emptyset$.

Główne narzędzie jakim będziemy operowali stanowi para:

- warstwowa rodzina Ω podzbiorów, o nieparzystej mocy, zbioru V
- funkcja $\pi : \Omega \rightarrow \mathbb{Q}$ spełniająca następujące warunki:

(i) $\pi(U) \geq 0$ jeżeli $U \in \Omega$ oraz $|U| \geq 3$

(ii) $\sum_{U \in \Omega, e \in \delta(U)} \pi(U) \leq w(e)$ dla każdego $e \in E$, gdzie $\delta(U)$ oznacza zbiór krawędzi incydentnych z U

Zauważmy, że powyższe dwa warunki implikują, że jeżeli M jest dowolnym skojarzeniem doskonałym w G to

$$w(M) \geq \sum_{U \in \Omega} \pi(U),$$

ponieważ

$$\begin{aligned} w(M) &= \sum_{e \in M} w(e) \geq \sum_{e \in M} \sum_{U \in \Omega, e \in \delta(U)} \pi(U) = \\ &= \sum_{U \in \Omega} \pi(U) |M \cap \delta(U)| \geq \sum_{U \in \Omega} \pi(U) \end{aligned}$$

Zatem M jest skojarzeniem doskonałym o minimalnej wadze jeżeli mamy w powyższym wzorze wszędzie równości zamiast nierówności.

Oznaczenia i założenia:

- dla każdej krawędzi e zdefiniujemy

$$w_\pi(e) := \sum_{U \in \Omega, e \in \delta(U)} \pi(U)$$

Oznacza to, że $w_\pi(e) \geq 0$, $e \in E$.

- E_π oznacza zbiór tych krawędzi e dla których $w_\pi(e) = 0$ i niech $G_\pi = (V, E_\pi)$
- w algorytmie zakładamy, że $\{v\} \in \Omega$, dla każdego $v \in V$
- ponieważ rodzina Ω jest warstwowa, dlatego rodzina Ω^{max} , złożona z tych zbiorów z Ω które są maksymalne w sensie zawierania (nie są zawarte w żadnym innym zbiorze), stanowi podział zbioru V

- przez G' oznaczmy graf otrzymany z G_π poprzez “ściągnięcie” wszystkich zbiorów z Ω^{max} :
 $G' = G_\pi \circ \Omega^{max}$ - oznacza to, że G' zależy od Ω oraz π
- zauważmy, że zbiór wierzchołków grafu G' stanowią zbiory z rodziny Ω^{max} i dwa wierzchołki, powiedzmy $U, U' \in \Omega^{max}$ są przyległe w G' gdy G_π ma krawędź łączącą pewien element ze zbioru U z pewnym elementem ze zbioru U'
- dla $U \in \Omega$, $|U| \geq 3$, oznaczamy przez H_U graf otrzymany z podgrafu indukowanego przez zbiór U w grafie G_π , $(G_\pi[U])$, poprzez “ściągnięcie” wszystkich maksymalnych ze względu na zawieranie właściwych podzbiorów zbioru U , które należą do Ω

Przechowujemy:

- warstwową rodzinę Ω “nieparzystych” podzbiorów zbioru wierzchołków V
- funkcję $\pi : \Omega \rightarrow \mathbb{Q}$
- skojarzenie M
- dla każdego $U \in \Omega$, $|U| \geq 3$, cykl C_U przechodzący przez wszystkie wierzchołki grafu H_U

Algorytm znajdowania skojarzenia doskonałego o minimalnej wadze

Dane: ważony graf prosty $G = (V, E)$, zawierający co najmniej jedno skojarzenie doskonałe

Poszukiwane: skojarzenie doskonałe o najmniejszej wadze w G

- $\Omega \leftarrow \{\{v\} : v \in V\}$, $M \leftarrow \emptyset$, $\pi(\{v\}) \leftarrow 0$, dla każdego $v \in V$
- niech X będzie zbiorem wierzchołków grafu G' nienasyconych przez M
- przyjmujemy, że ścieżka ma dodatnią długość gdy ma co najmniej jedną krawędź

1. G' zawiera M -przemienny $X - X$ spacer o dodatniej długości.

- Wybierz najkrótszy taki spacer P . Jeżeli P jest ścieżką to jest ścieżką M -powiększającą w G' .
 $M \leftarrow M \triangle E(P)$ i iteruj.
- Jeżeli P nie jest ścieżką to zawiera M -kwiat (patrz odpowiednie twierdzenie z paragrafu 7.4). Niech C będzie kwieciem (cyklem) tego kwiatu. Dodaj $U \leftarrow \bigcup V(C)$ do rodziny Ω ("ściąganie"), podstaw $\pi(U) \leftarrow 0, M \leftarrow M \setminus E(C), C_U \leftarrow C$ i iteruj.

2. G' nie zawiera M -przemiennego $X - X$ spaceru o dodatniej długości.

- Niech \mathcal{S} będzie zbiorem wierzchołków U grafu G' dla którego G' ma M -przemienny $X - U$ spacer nieparzystej długości i niech \mathcal{T} będzie zbiorem wierzchołków U grafu G' dla którego G' ma M -przemienny $X - U$ spacer parzystej długości.
- $\pi(U) \leftarrow \pi(U) + \alpha$ jeżeli $U \in \mathcal{T}$ oraz $\pi(U) \leftarrow \pi(U) - \alpha$ jeżeli $U \in \mathcal{S}$, gdzie α jest największą wartością spełniającą warunki (i) oraz (ii) dla funkcji π .
- Jeżeli $\pi(U) = 0$ dla pewnego $U \in \mathcal{S}$, gdzie $|U| \geq 3$, to usuń U z Ω ("rozciąganie"), roszerz M o skojarzenie doskonałe zawarte w $C_U - v$, gdzie v jest wierzchołkiem M -nasyconym, i iteruj.

- Proces iteracyjny kończy się jeżeli M jest skojarzeniem doskonałym w G' , wtedy używając C_U możemy rozszerzyć M do skojarzenia doskonałego N w G takiego, że $w_\pi(N) = 0$ i $|N \cap \delta(U)| = 1$ dla każdego $U \in \Omega$. Zatem dla N mamy wszędzie równości wyjściowym zbiorze, co oznacza, że N jest skojarzeniem o najmniejszej wadze w G

Twierdzenie . Algorytm znajduje skojarzenie doskonałe o minimalnej wadze w czasie $O(n^2m)$.

Wniosek . W dowolnym ważonym grafie można znaleźć skojarzenie o największej wadze w czasie $O(n^2m)$.

Dowód. Niech $G = (V, E)$ będzie grafem z funkcją ważenia w . Utwórz nowy graf w następujący sposób: weź kopie G' oraz w' , odpowiednio, G oraz w . Połącz krawędzią każdy wierzchołek $v \in V$ z jego kopią w G' i przyporządkuj im wagę 0. Jeżeli M jest skojarzeniem doskonałym o maksymalnej wadze w nowym grafie, to ograniczenie tego skojarzenia do krawędzi wyjściowego grafu G jest skojarzeniem o największej wadze w tym grafie. ■

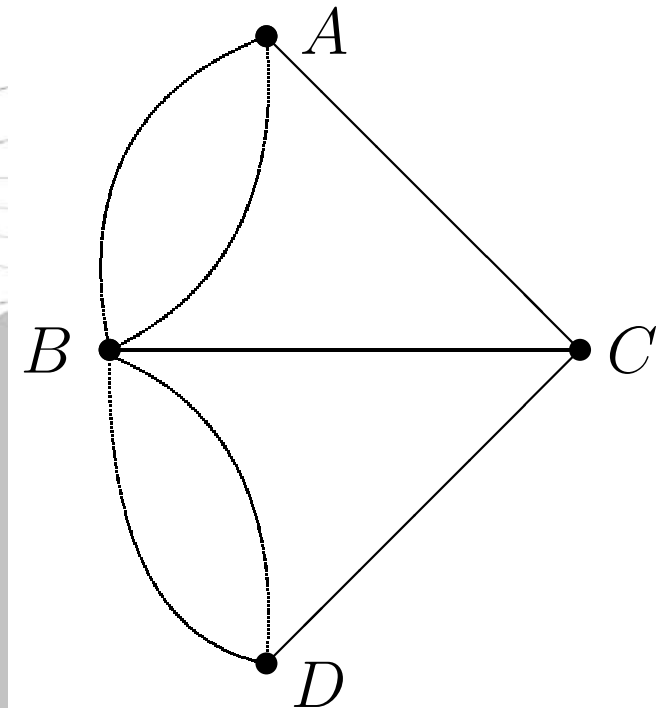
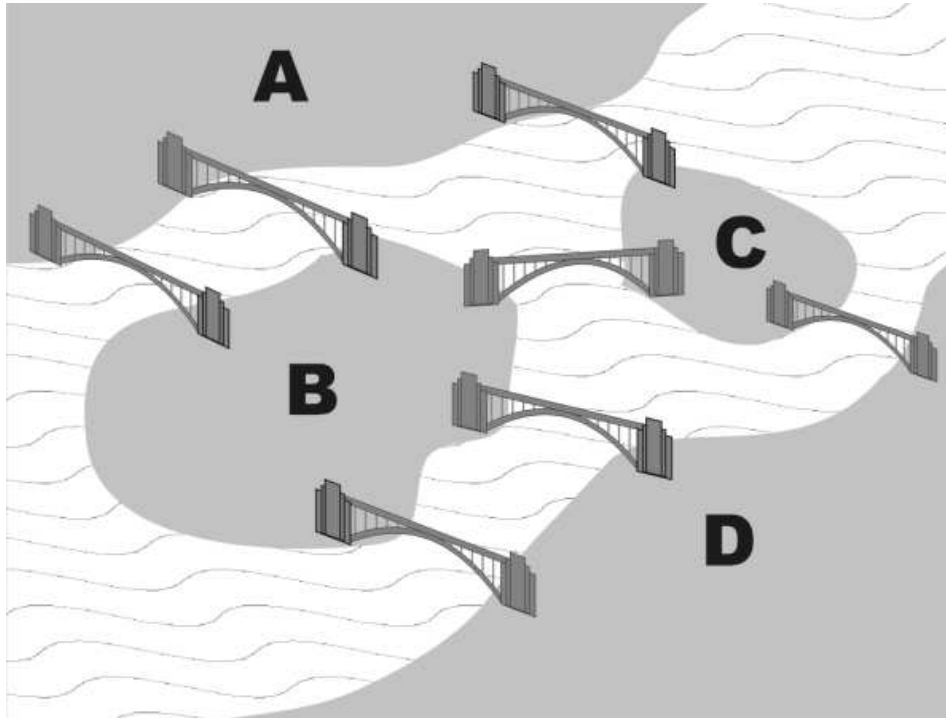
8. Obchód Eulera grafu

8.1 Definicje i twierdzenia

Definicja (Szlak i obchód Eulera). *Jeżeli w spacerze wszystkie krawędzie są różne to taki spacer nazywamy szlakiem. Szlak, który zawiera każdą krawędź $e \in E(G)$ nazywamy szlakiem Eulera grafu G . Obchód Eulera grafu jest to domknięty szlak Eulera, tzn. szlak rozpoczynający i kończący się w tym samym wierzchołku.*

Definicja (Graf eulerowski i półeulerowski). *Graf jest eulerowski jeżeli zawiera obchód Eulera, a półeulerowski jeżeli zawiera szlak Eulera.*

Mosty w Królewcu roku 1736



Czy można zrobić spacer po Królewcu tak, aby przez każdy most przejść dokładnie jeden raz i wrócić do punku wyjścia?

- potrafimy podać warunek dostateczny i konieczny na istnienie szlaku i obchodu Eulera oraz zdefiniować efektywne algorytmy je wyznaczające
- warunek przypisywany jest L. Eulerowi (1736) chociaż pierwszy pełny dowód opublikował w 1873 roku Carl Hierholzer (1840–1871).

Twierdzenie (Eulera–Hierholzera). *Niepusty spójny graf jest eulerowski wtedy i tylko wtedy, gdy nie posiada wierzchołków o nieparzystym stopniu.*

Wniosek . *Graf spójny ma szlak Eulera wtedy i tylko wtedy, gdy ma co najwyżej dwa wierzchołki stopnia nieparzystego.*

Dowód. (konieczność i wniosek)

- Niech graf G będzie eulerowski i niech C będzie obchodem Eulera grafu G o początku (i końcu) w wierzchołku u . Za każdym razem gdy wierzchołek v wystąpi jako wierzchołek wewnętrzny obchodu C , dwie krawędzie incydentne z v są włączone do C . Ponieważ obchód Eulera zawiera każdą krawędź grafu G dokładnie jeden raz, więc $d_G(v)$ jest parzysty dla wszystkich $v \in V(G) \setminus \{u\}$. Podobnie, ponieważ C rozpoczyna i kończy się w wierzchołku u , $d_G(u)$ jest również parzysty. Zatem G nie posiada żadnego wierzchołka stopnia nieparzystego.
- Zauważmy, że jeżeli graf ma co najwyżej dwa wierzchołki stopnia nieparzystego to znaczy, że nie ma ich wcale lub ma dokładnie dwa takie wierzchołki. W drugim przypadku, połączmy te dwa wierzchołki nową krawędzią e i zauważmy, że graf $G + e$ posiada obchód Eulera C . Zatem $C - e$ jest szukanym szlakiem Eulera grafu G . ■

8.2 Algorytmy Hierholzera i Fleury'ego

Algorytm Hierholzera opiera się na obserwacji, że obchód Eulera (jeżeli istnieje) jest sumą krawędziowo rozłącznych cykli.

Twierdzenie . Jeżeli w grafie G nie ma wierzchołka nieparzystego stopnia, to istnieją krawędziowo rozłączne cykle C_1, C_2, \dots, C_m takie, że

$$E(G) = E(C_1) \cup E(C_2) \cup \dots \cup E(C_m).$$

Dowód. (indukcja ze względu na liczbę krawędzi grafu)

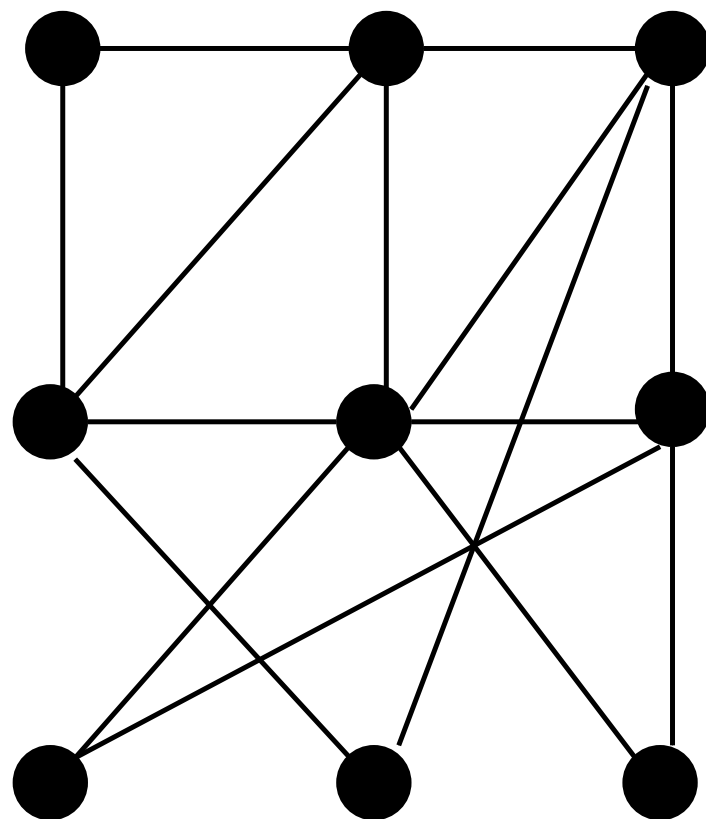
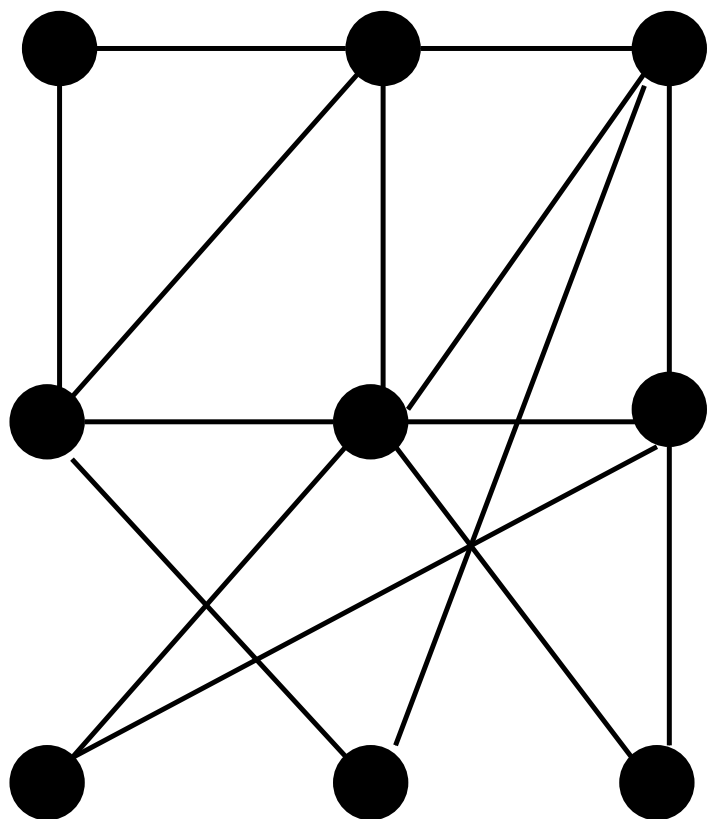
Jeżeli $E(G) = 1$ i w grafie G nie ma wierzchołka nieparzystego stopnia, to jedyna krawędź musi być pętlą i twierdzenie jest prawdziwe.

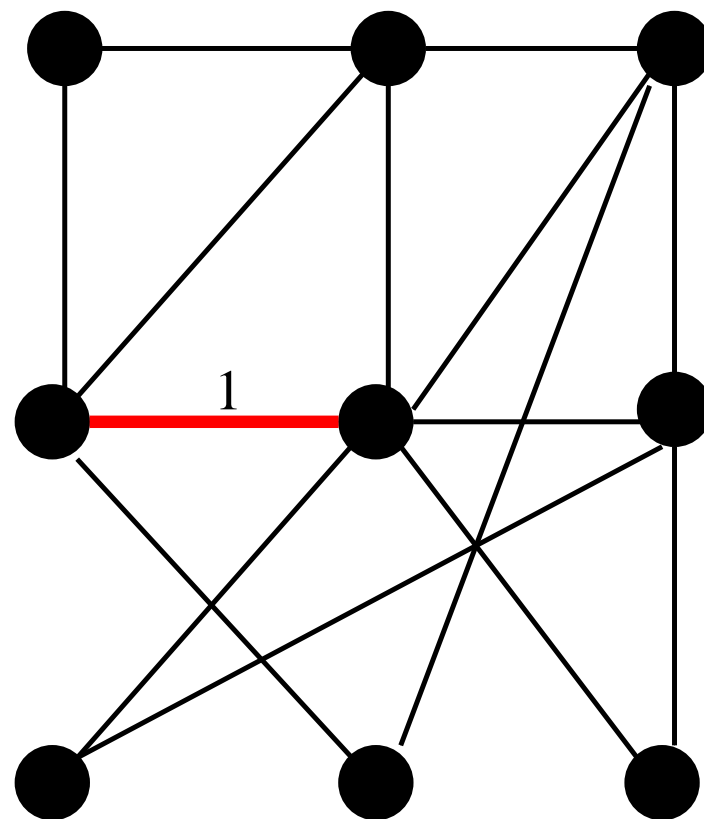
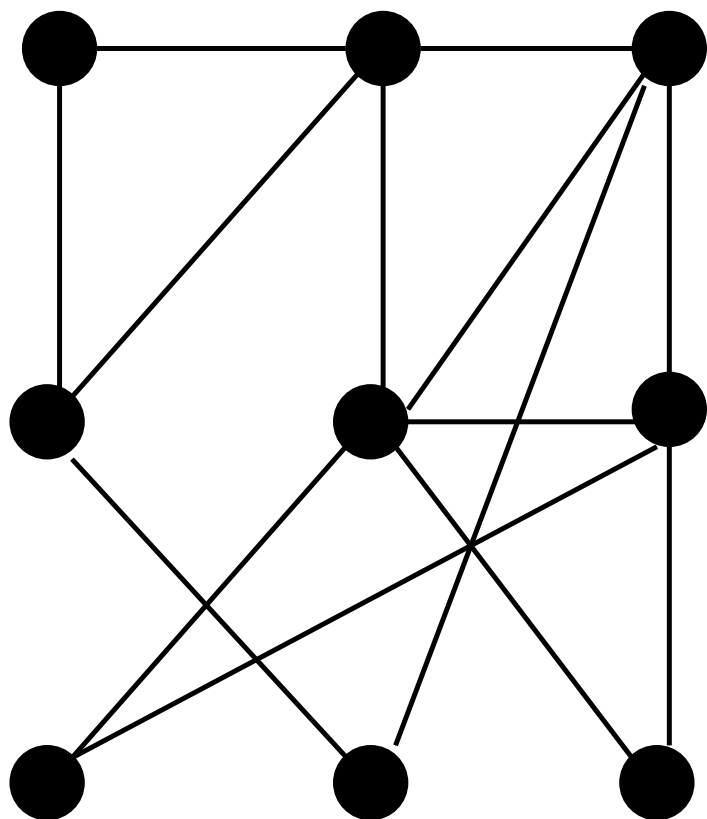
Założmy, że teza naszego twierdzenia jest prawdziwa dla wszystkich grafów o mniej niż $E(G)$ krawędziach i niech G będzie grafem o $E(G)$ krawędziach, w którym nie ma wierzchołka nieparzystego stopnia.

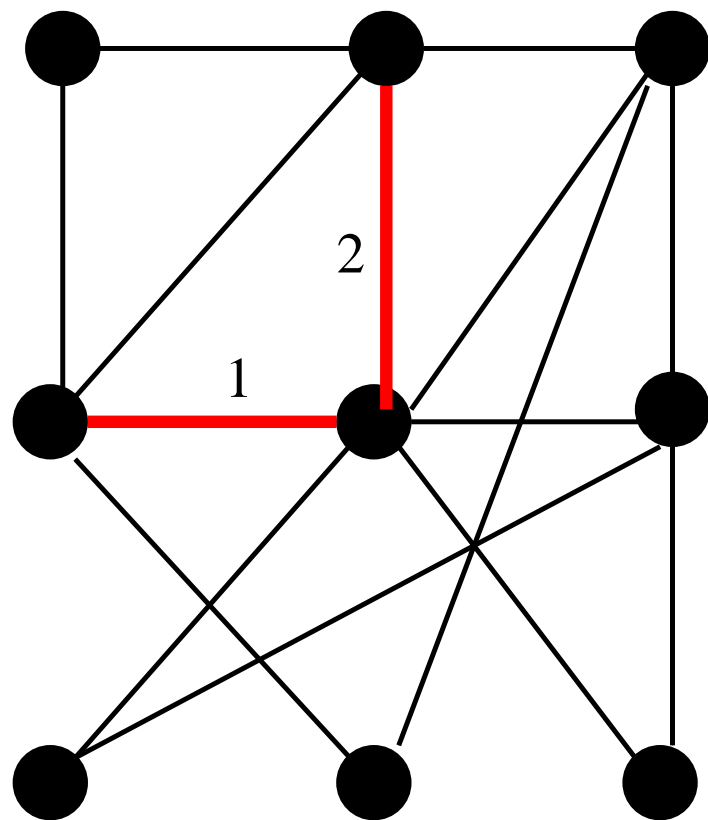
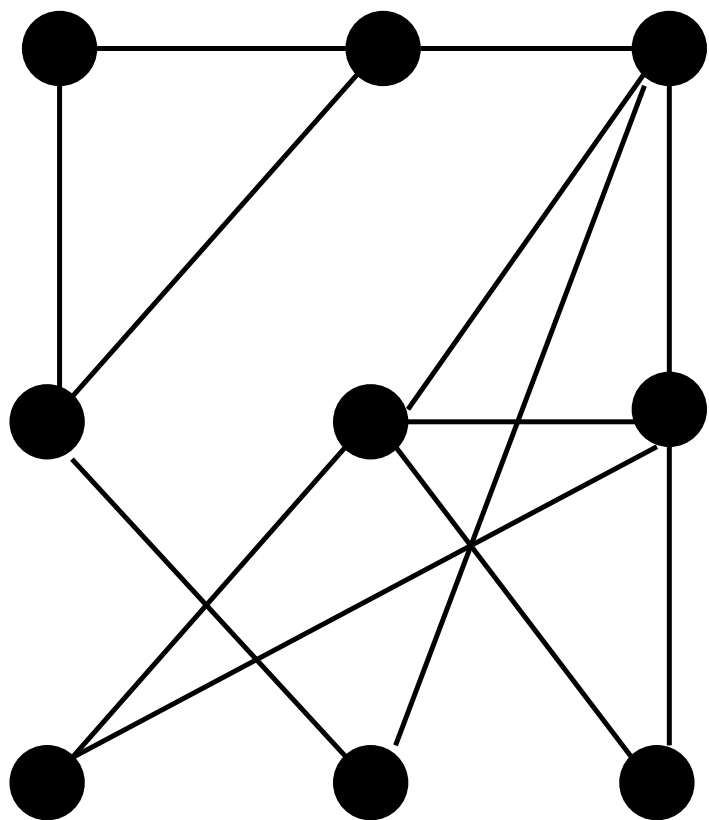
Jeżeli z G usuniemy wierzchołki izolowane, to w otrzymanym grafie minimalny stopień jest co najmniej równy 2. Wtedy łatwo zauważyć, że w G istnieje albo pętla, albo dwucykł, albo w przypadku grafu prostego, cykl długości większej niż 2. Oznaczmy ten cykl (pętlę, dwucykł) przez C i rozważmy graf G^* otrzymany z G poprzez usunięcie krawędzi cyklu C (tj. $G^* = G - E(C)$). Zauważmy, że G^* nie ma wierzchołków nieparzystego stopnia i oczywiście G^* ma mniej niż $E(G)$ krawędzi. Z założenia indukcyjnego wynika, że w G^* istnieje, powiedzmy, $m - 1$ krawędziowo rozłącznych cykli C_1, C_2, \dots, C_{m-1} takich, że $E(G^*) = \bigcup_{i=1}^{m-1} E(C_i)$. Przyjmując $C_m = C$, otrzymujemy tezę twierdzenia dla grafu G .

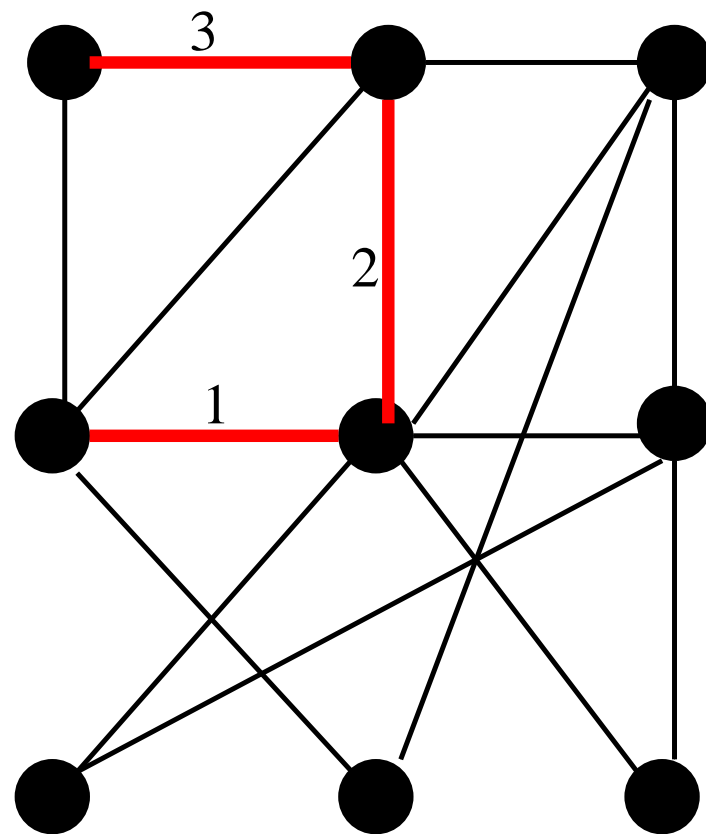
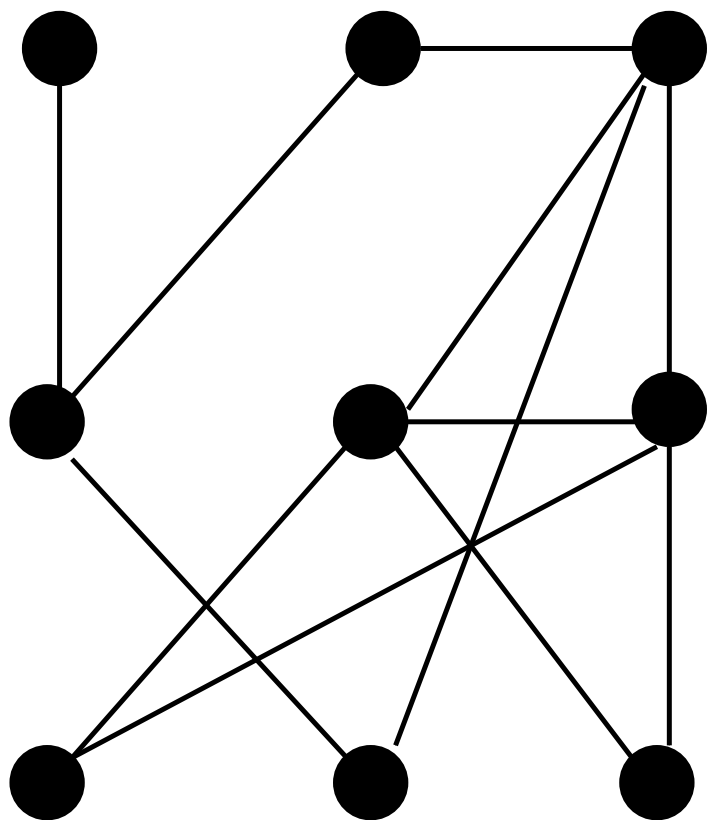
Algorytm Hierholzera

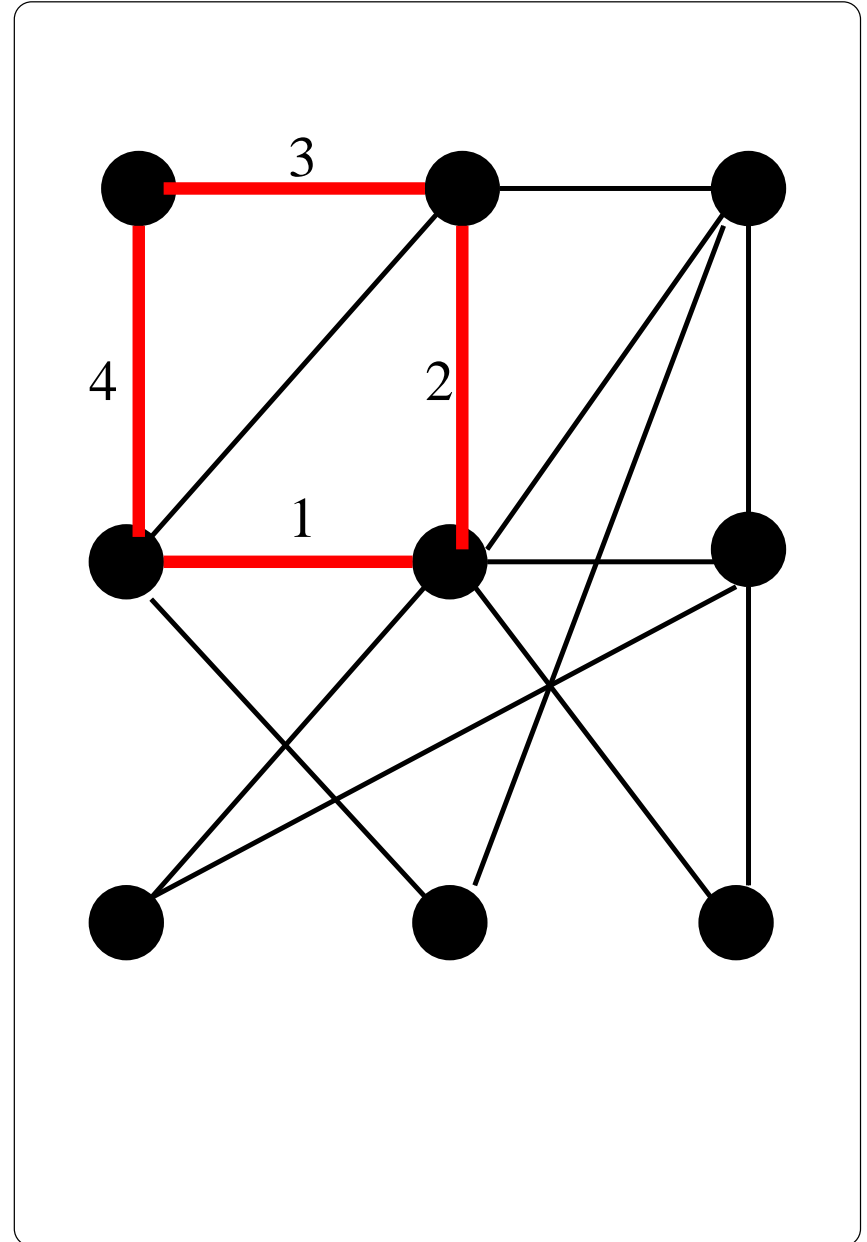
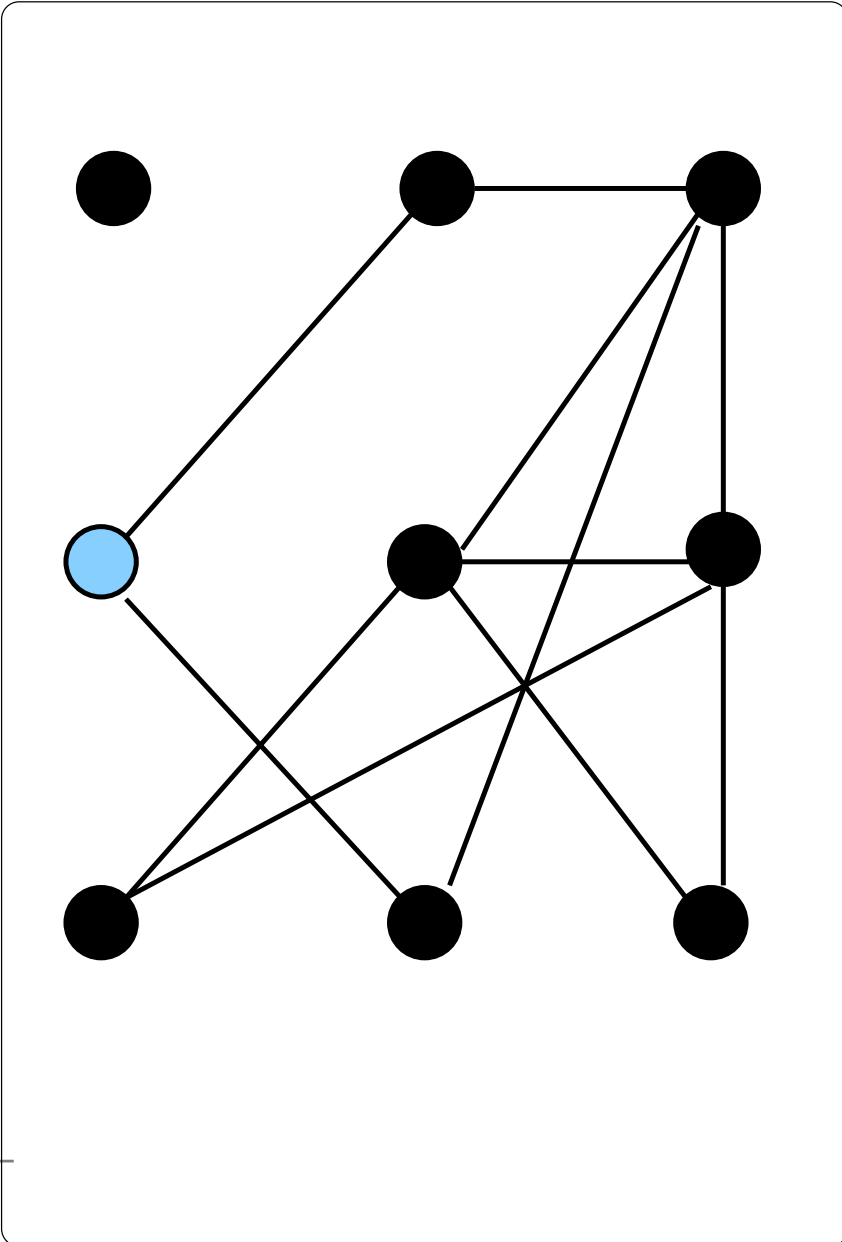
1. Rozpocznij od dowolnego wierzchołka $v \in V(G)$ i wyznacz dowolny szlak domknięty \mathcal{O} zaczynający i kończący się w v ; $G \leftarrow G - E(\mathcal{O})$.
2. Powtarzaj podpunkty (a), (b) i (c) tak długo, aż G będzie grafem pustym.
 - (a) Znajdź dowolny wierzchołek $w \in V(\mathcal{O})$ incydentny z pewną krawędzią grafu G .
 - (b) Znajdź w grafie G dowolny cykl \mathcal{C} zaczynający i kończący się w w .
 - (c) Połącz szlak domknięty \mathcal{O} z cyklem \mathcal{C} tworząc nowy szlak domknięty \mathcal{O} ; $G \leftarrow G - E(\mathcal{C})$.

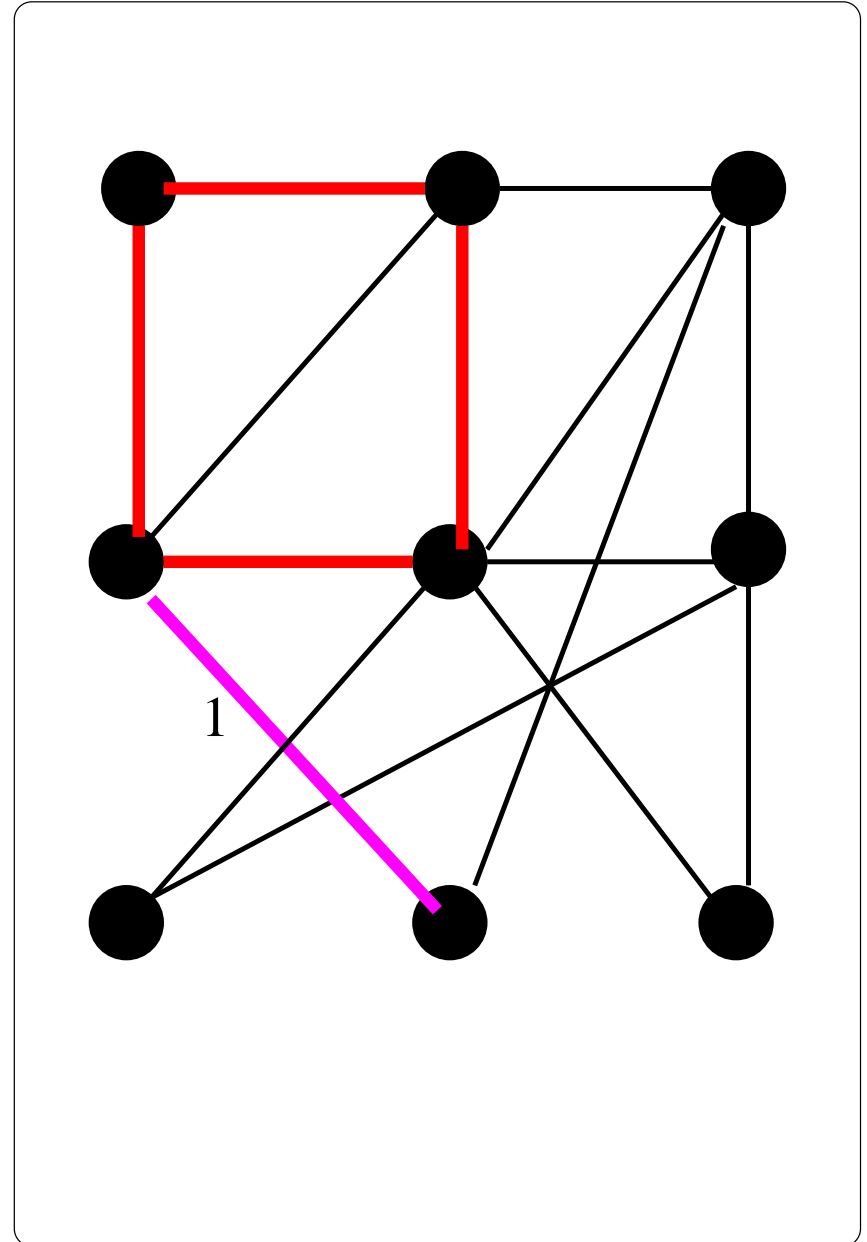
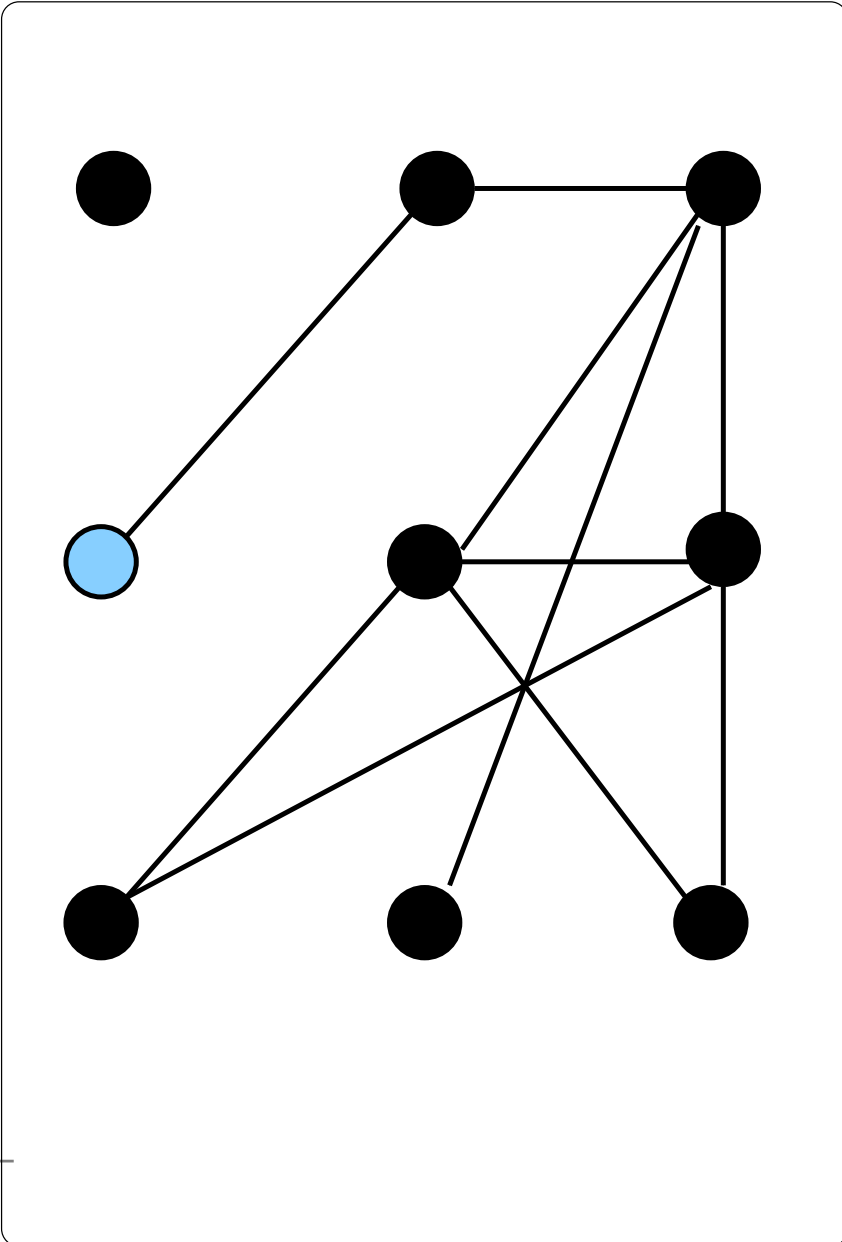


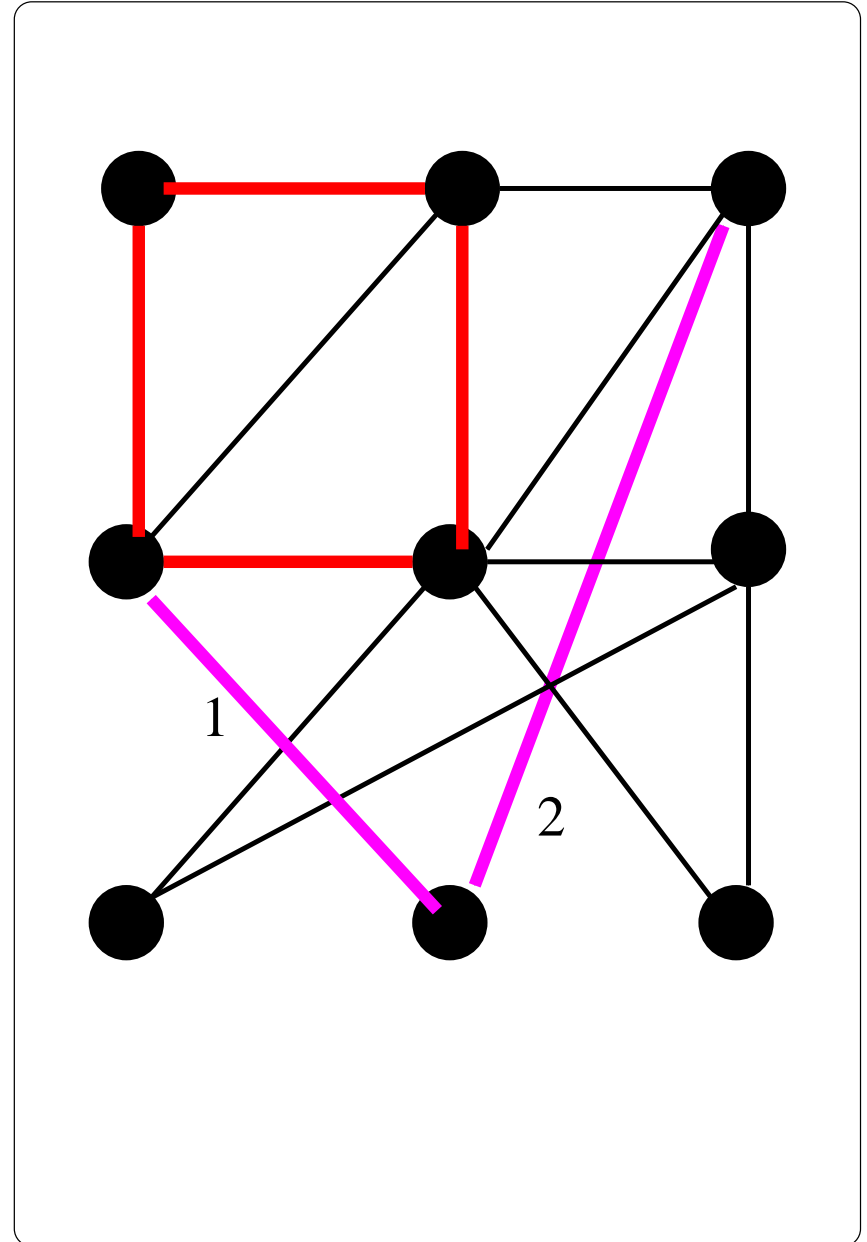
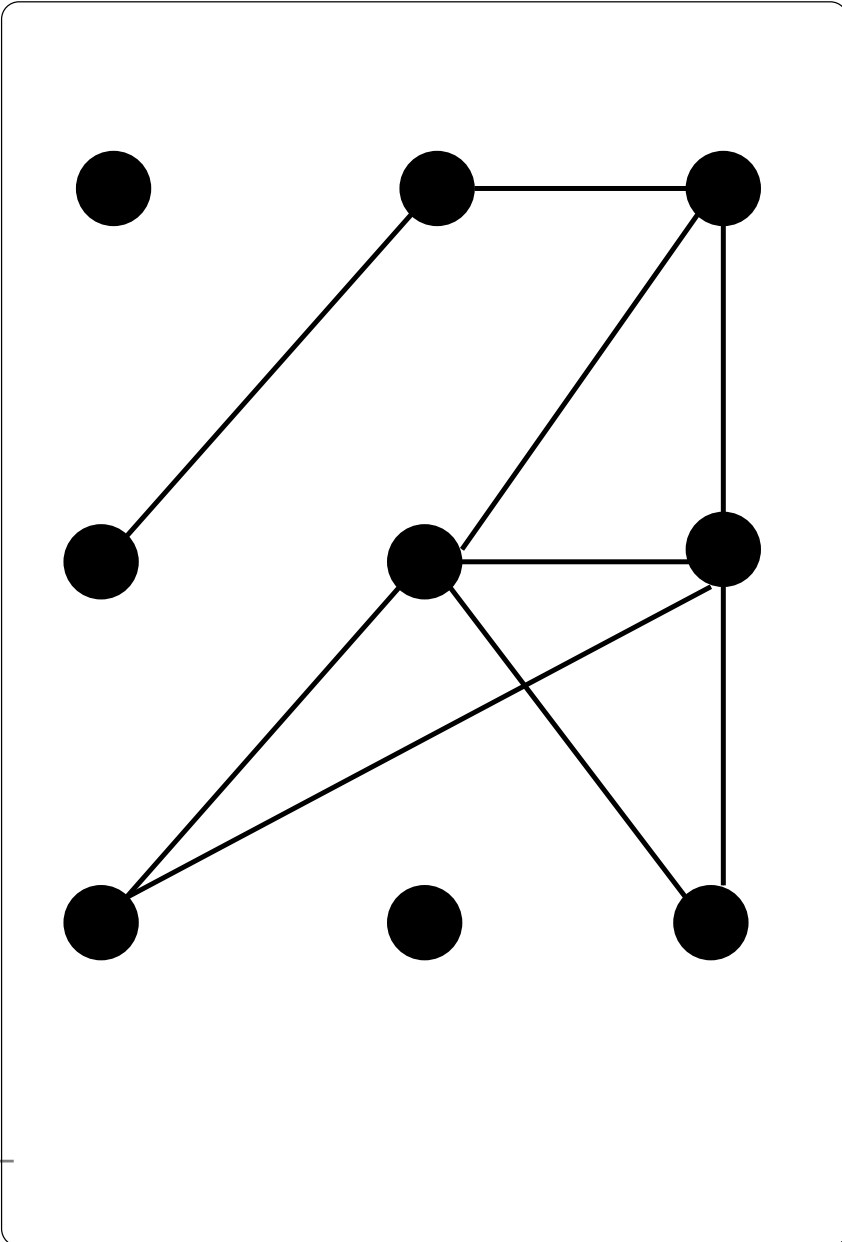


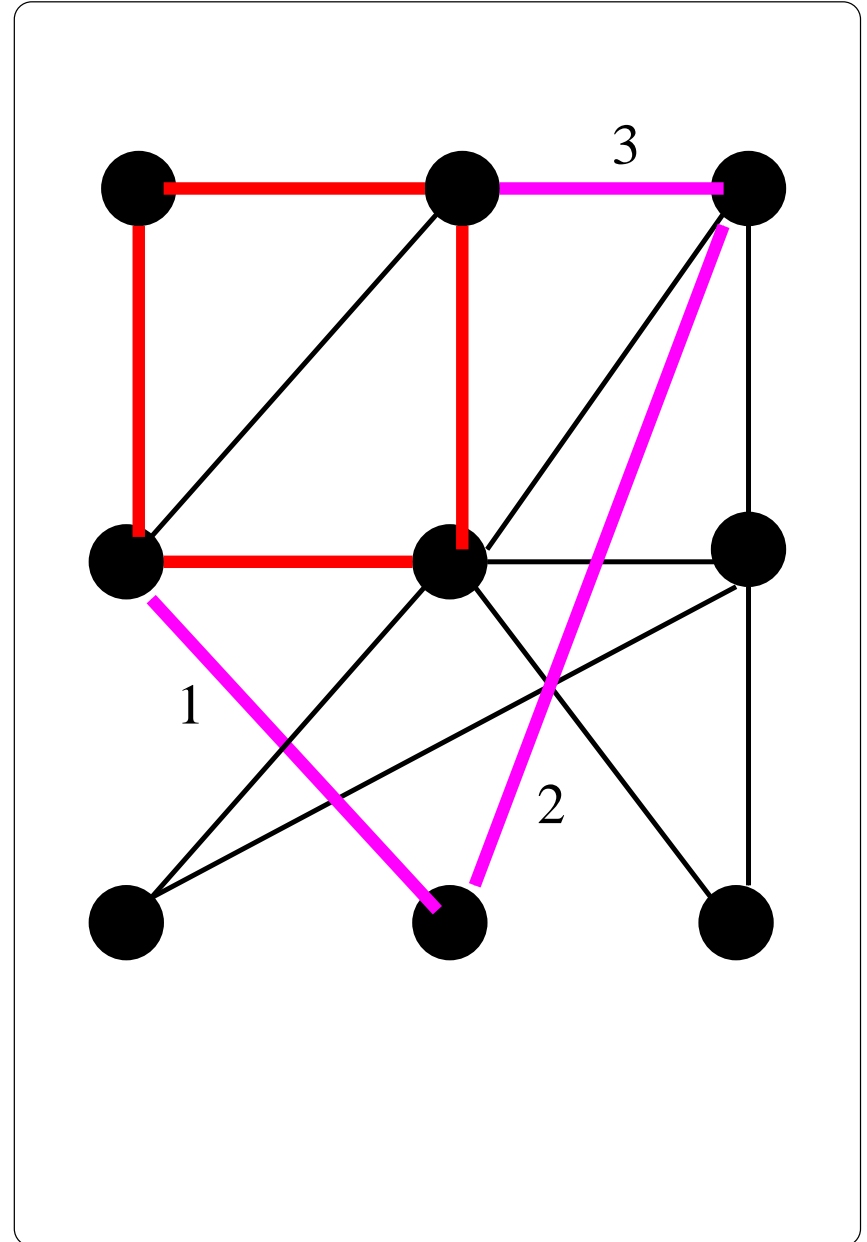
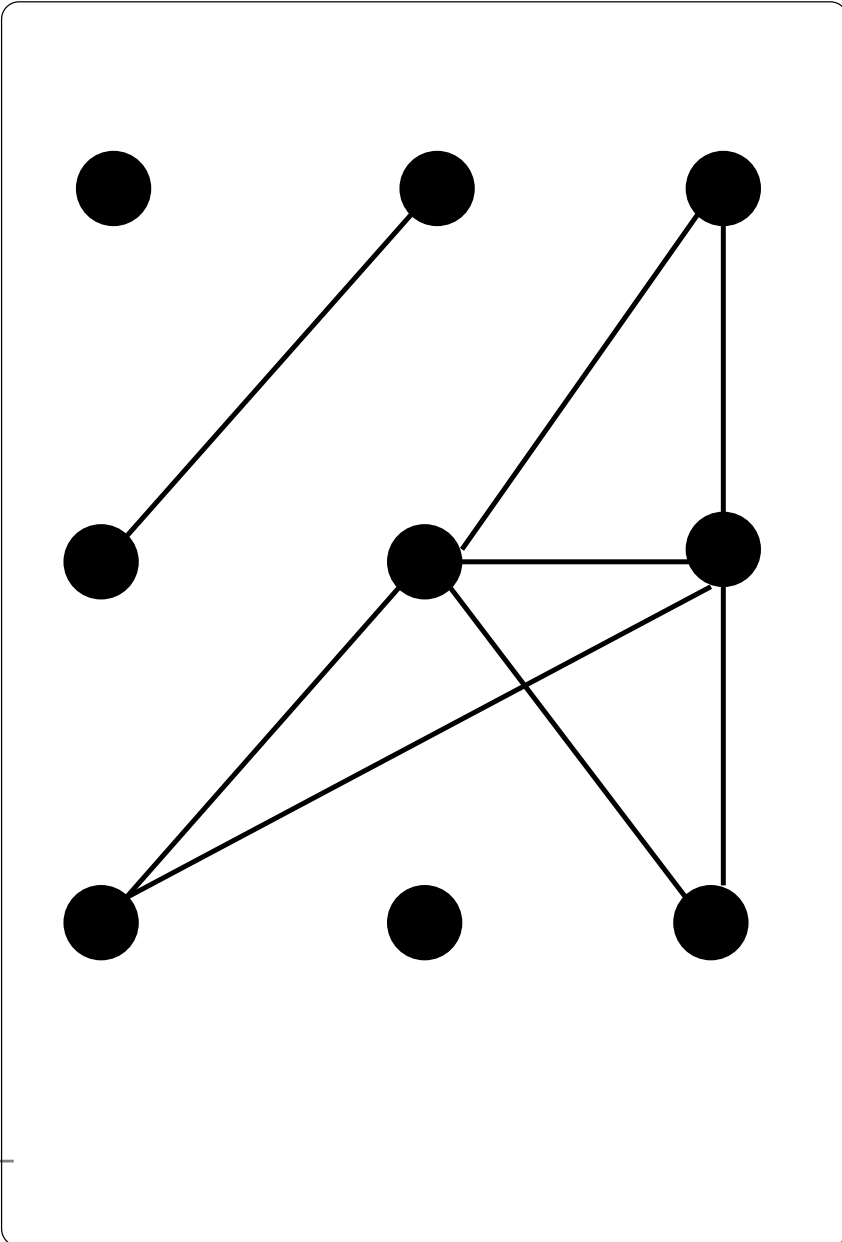


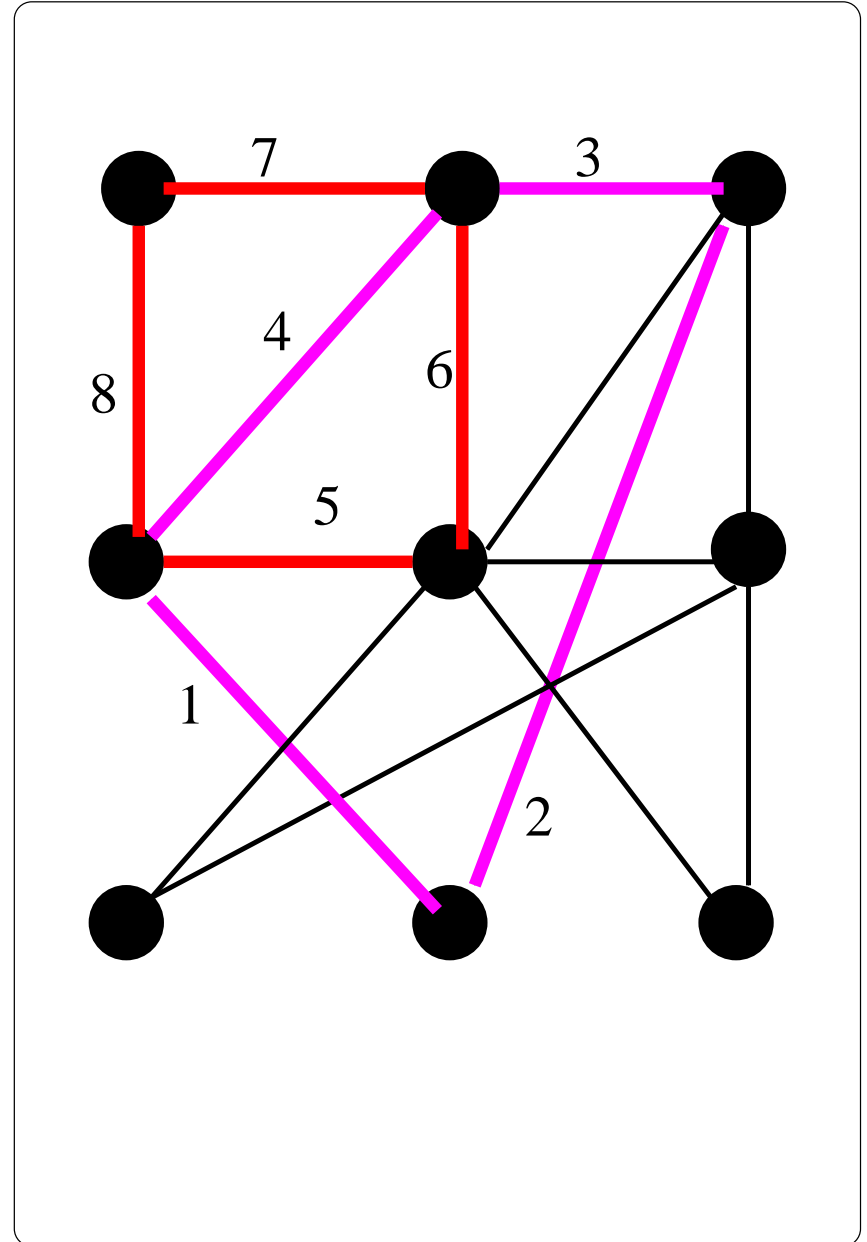
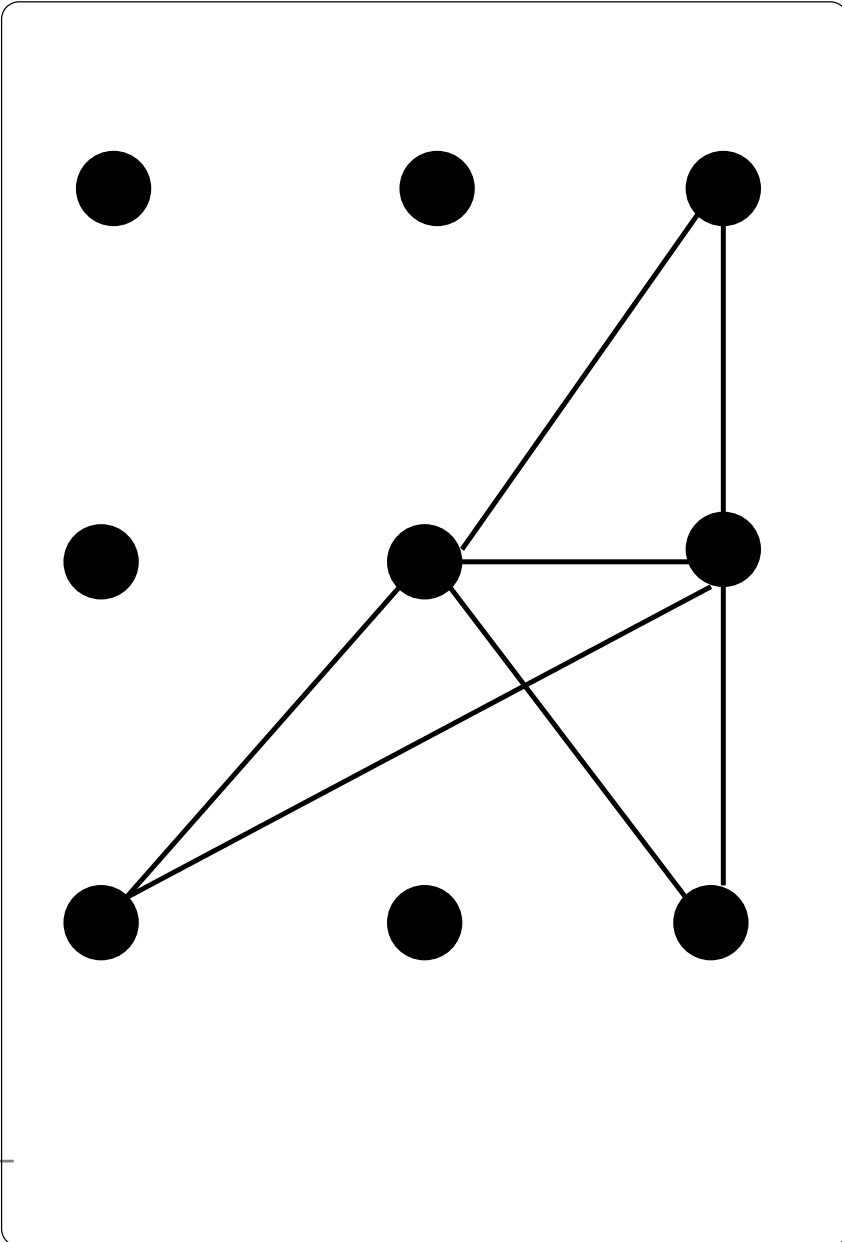


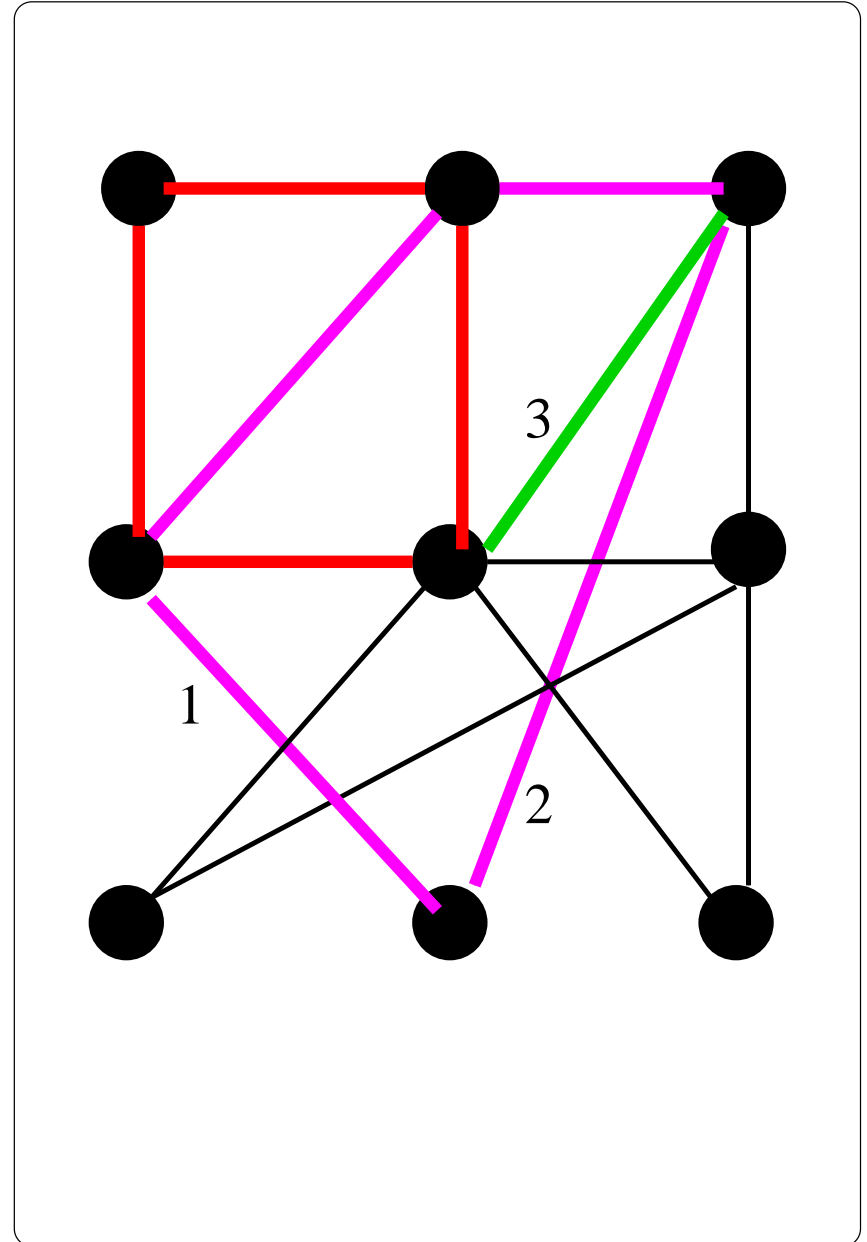
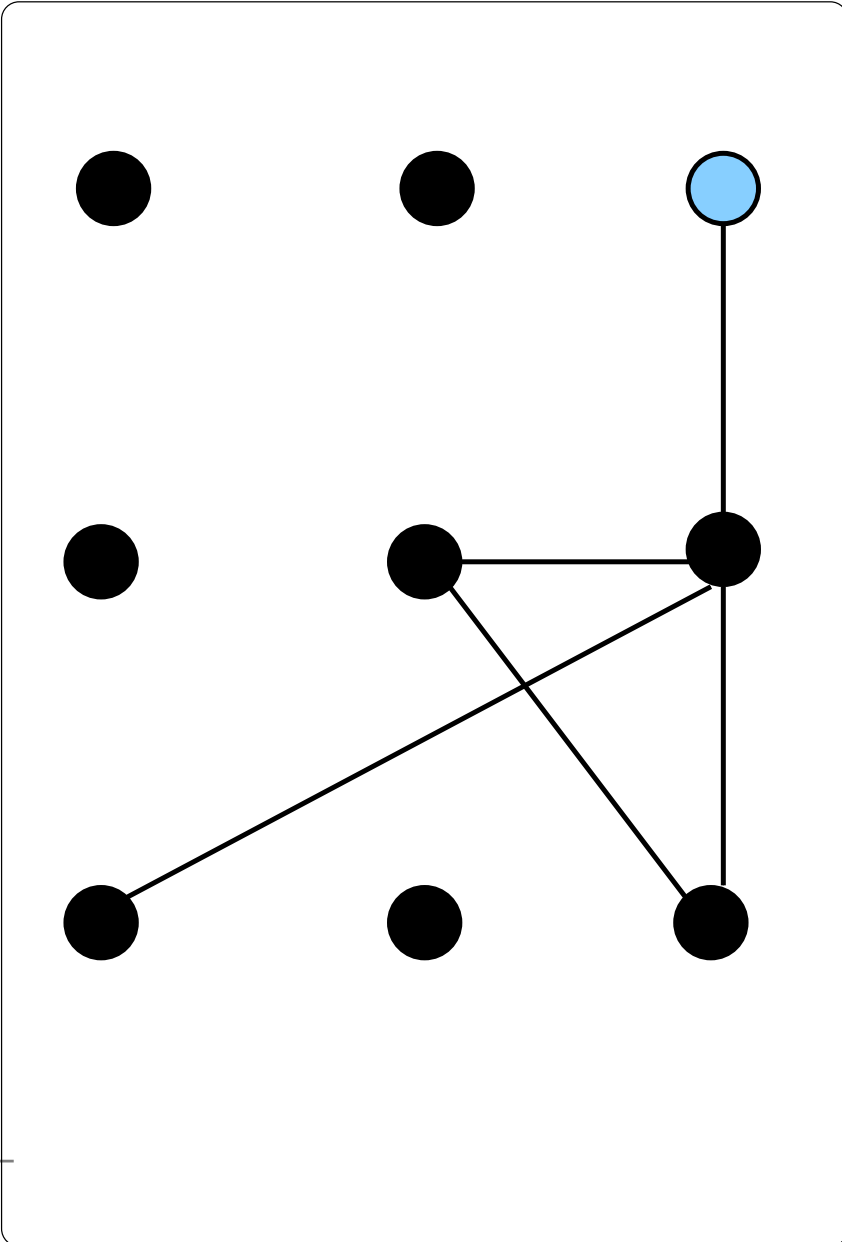


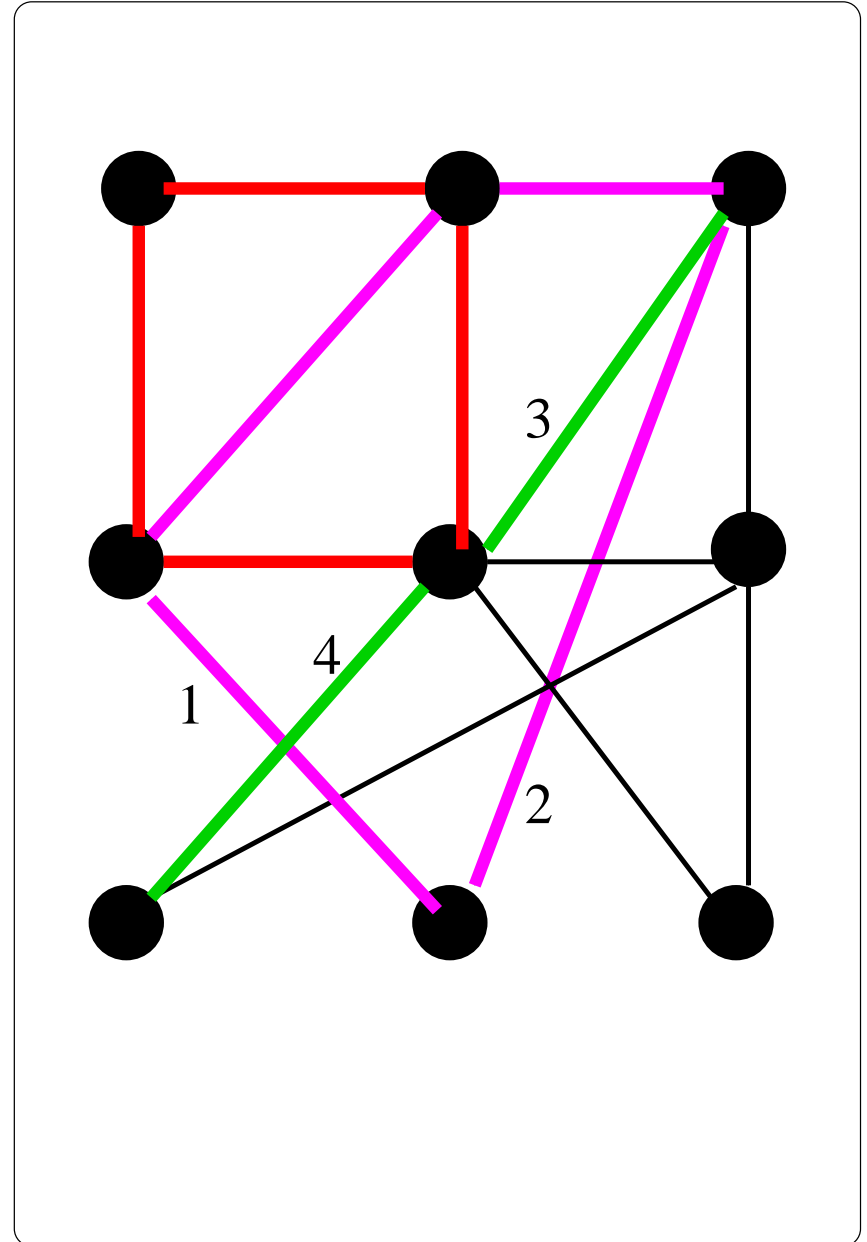
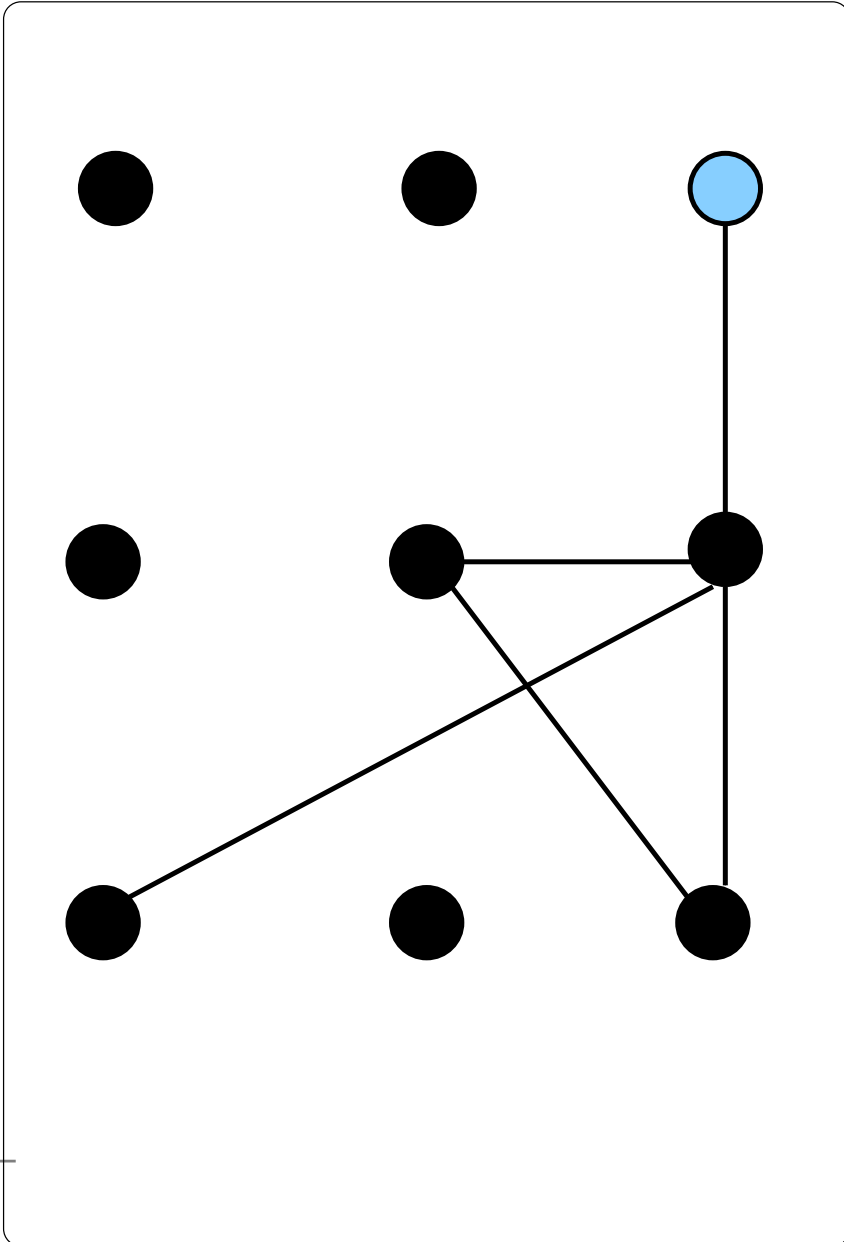


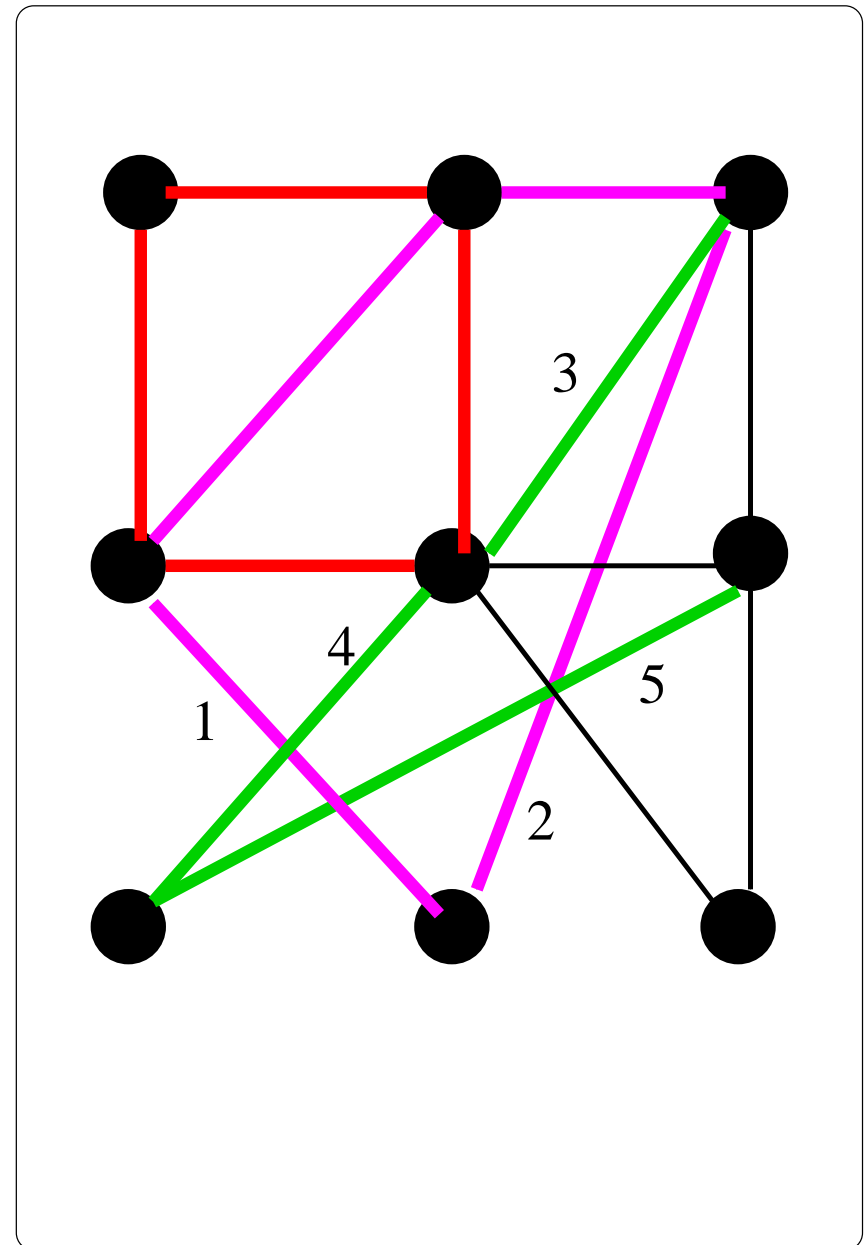
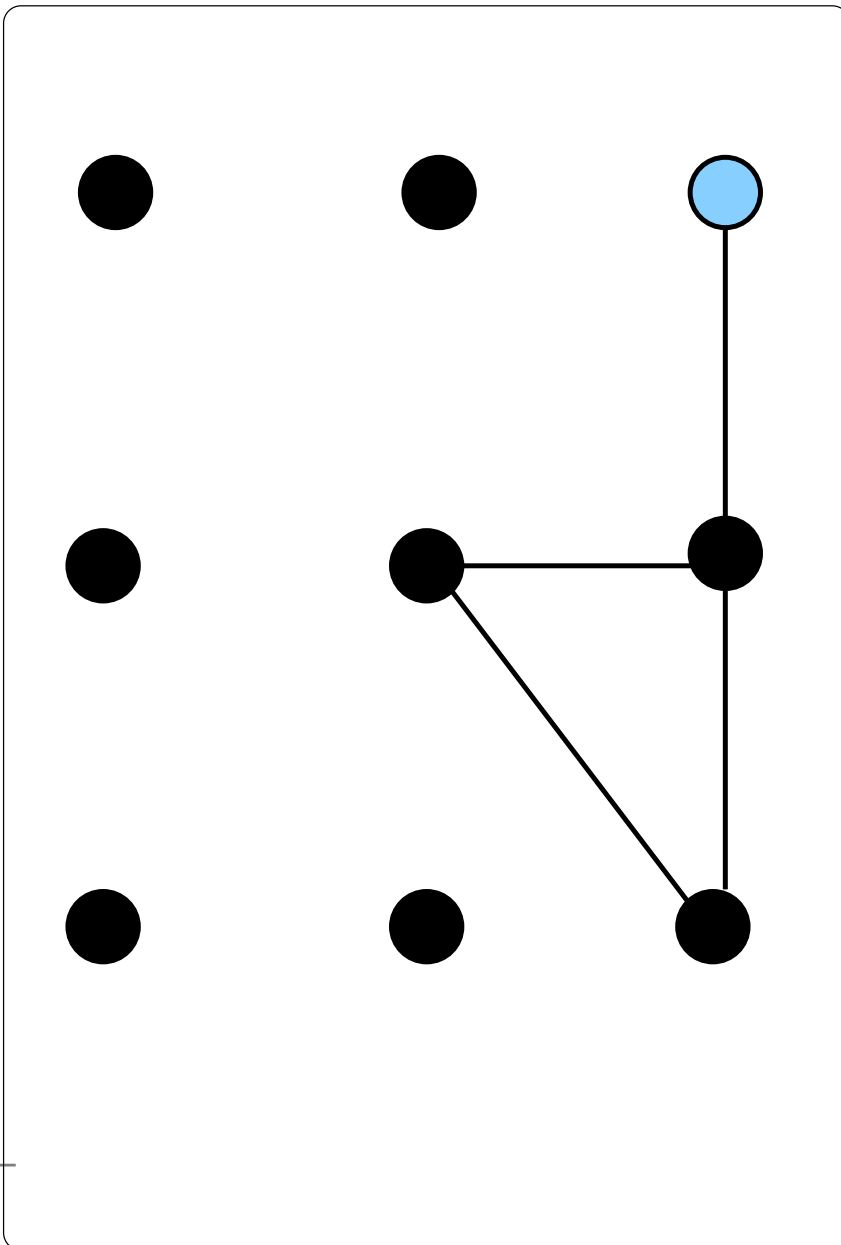


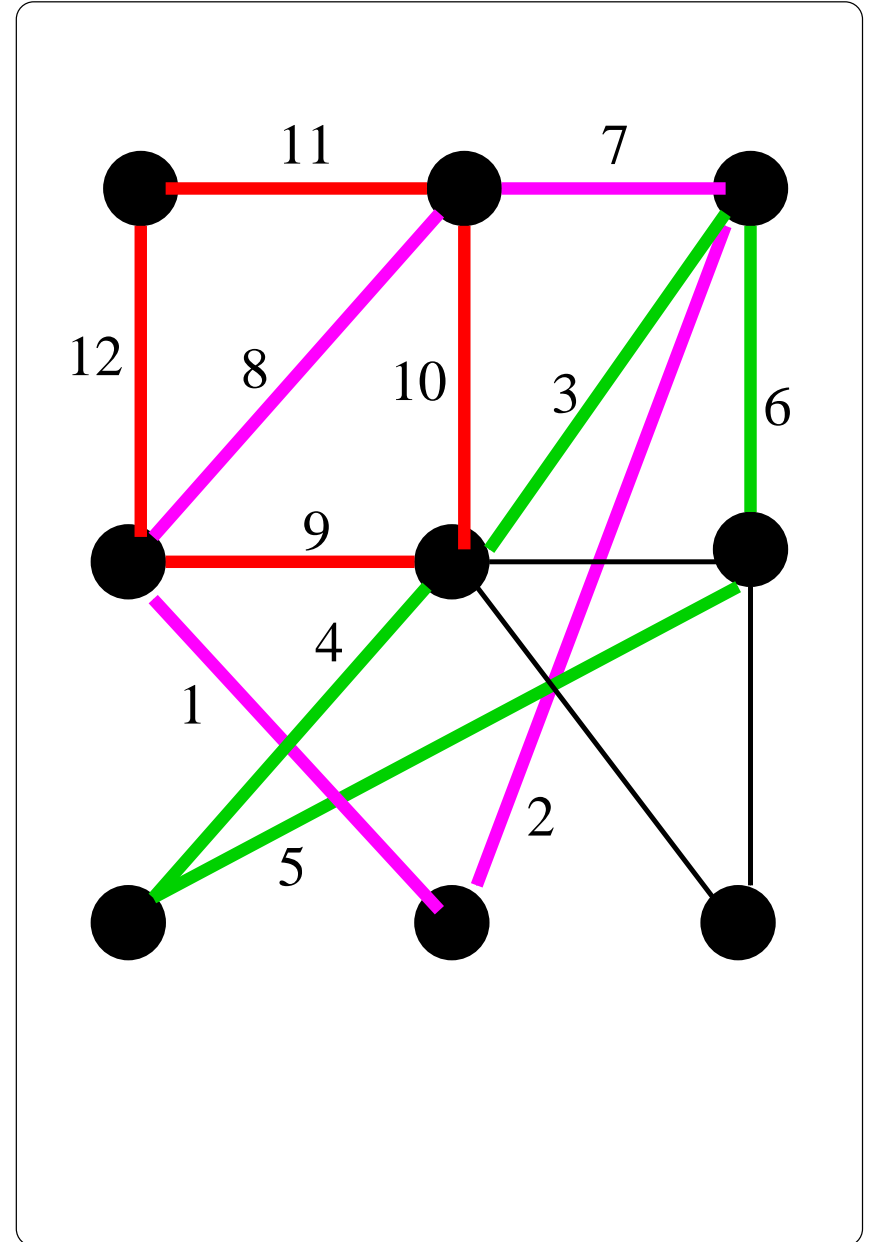
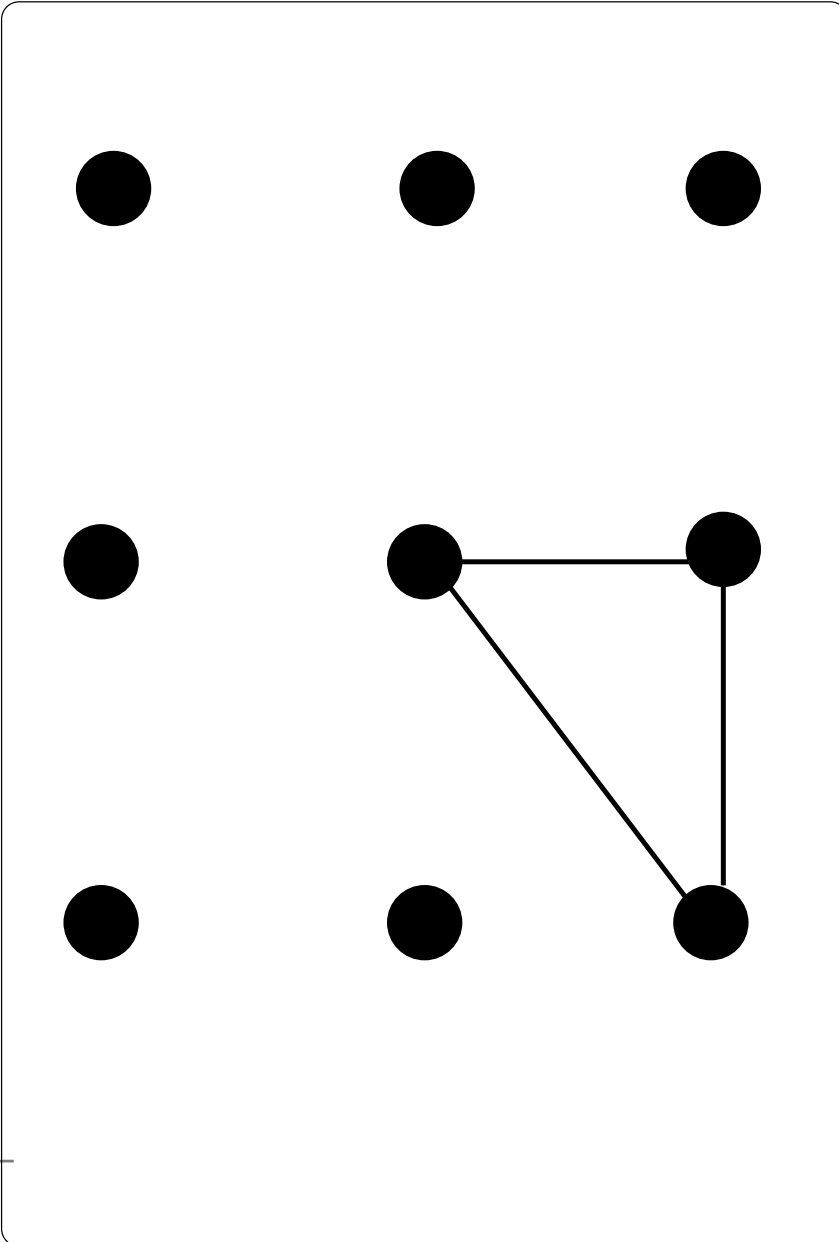


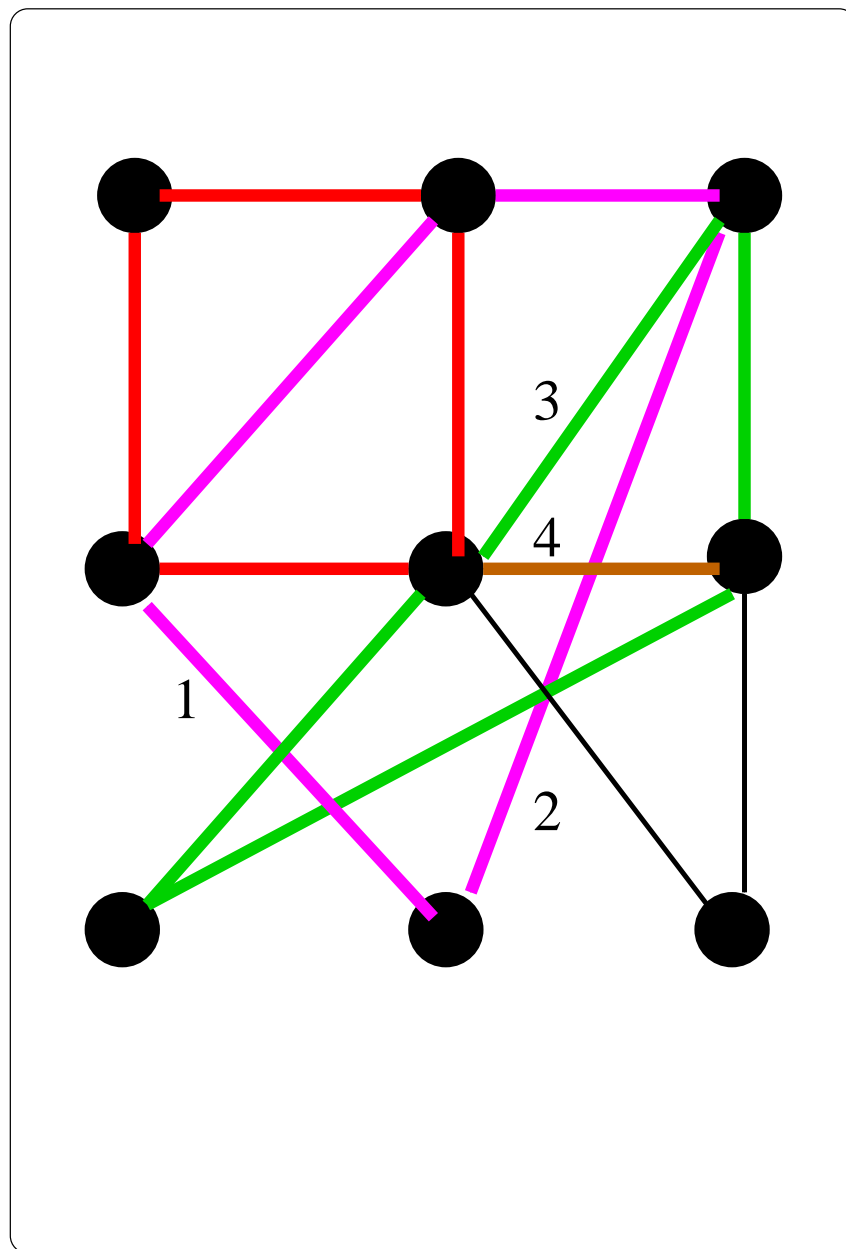
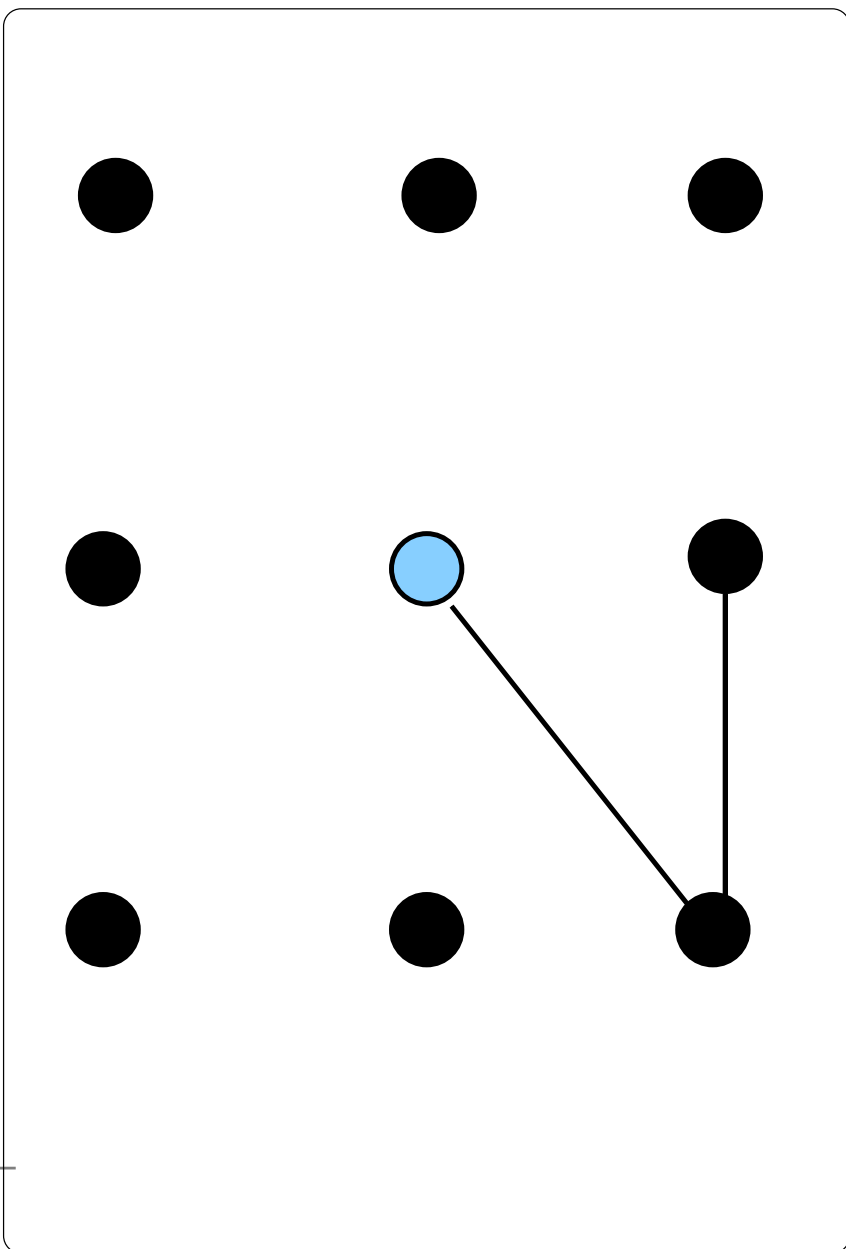


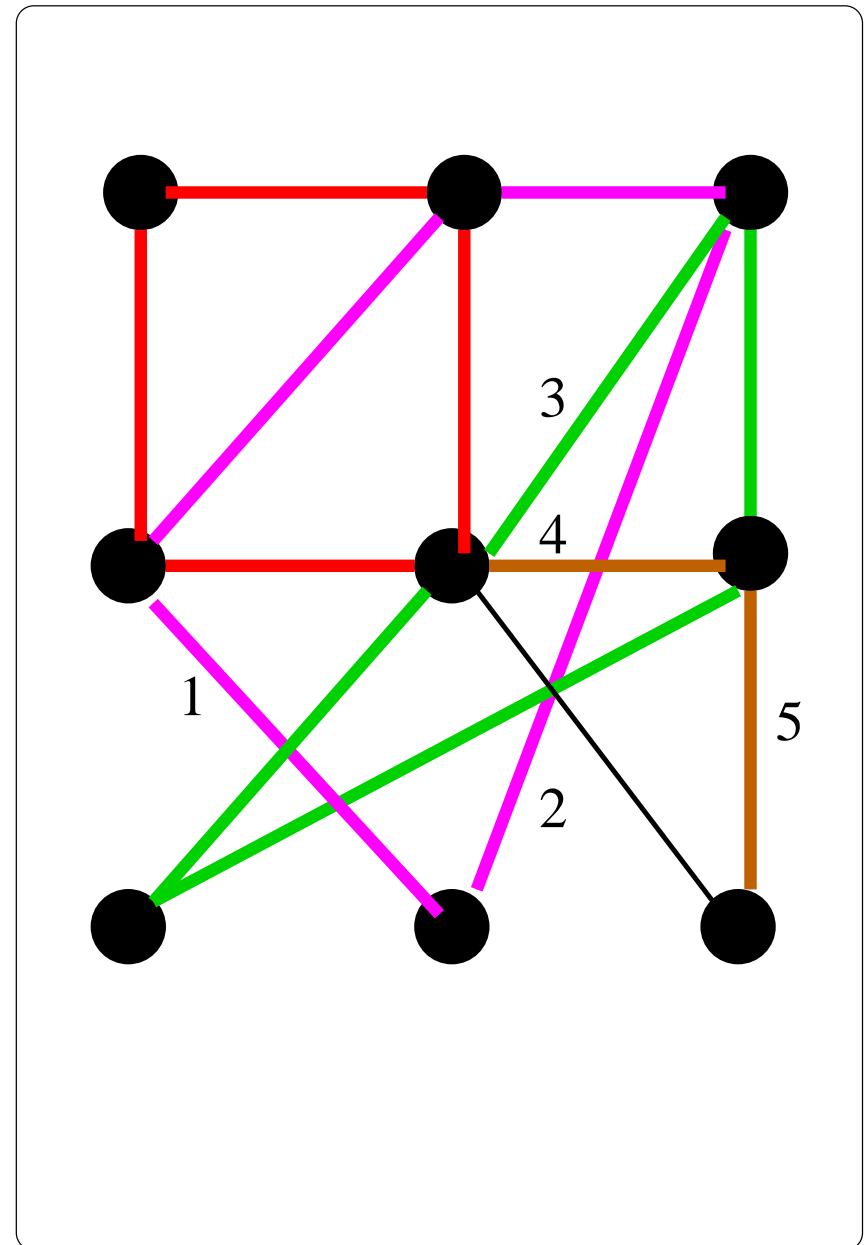
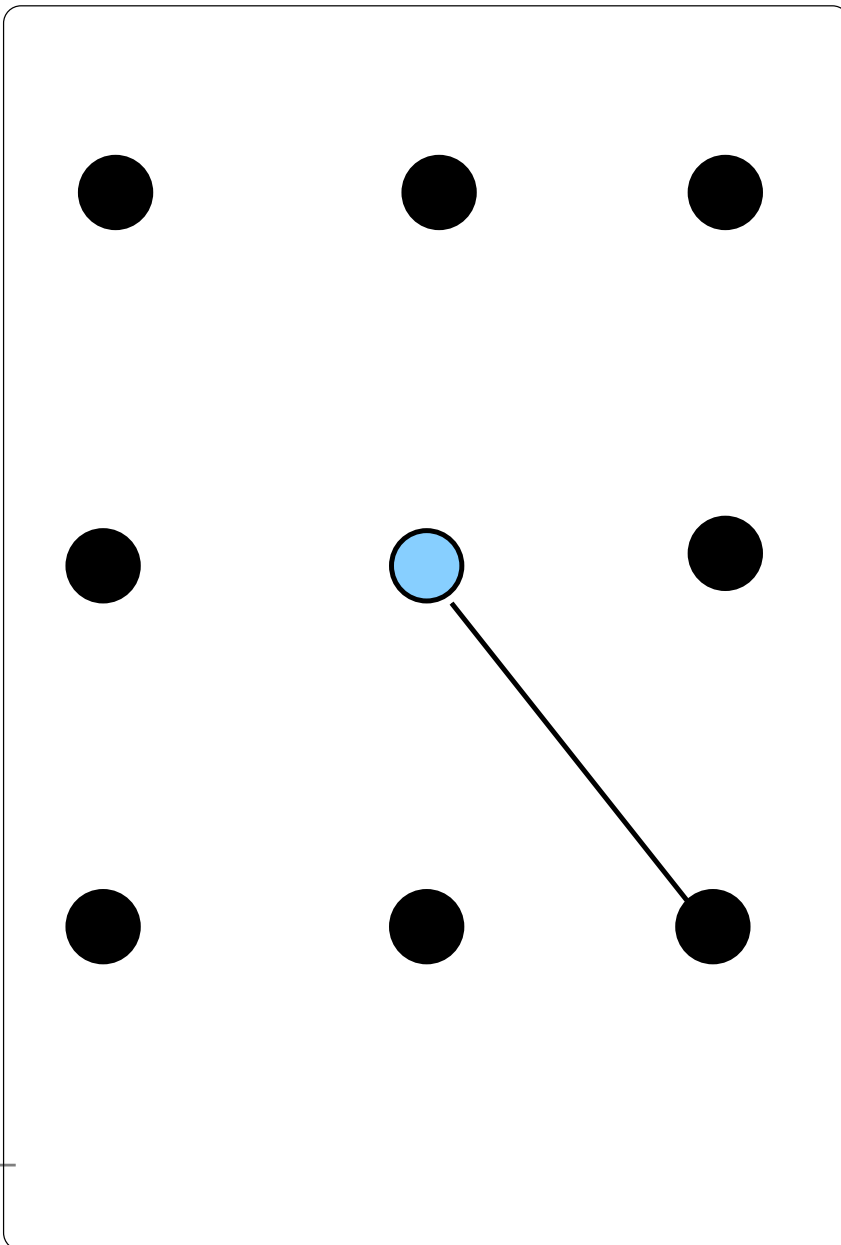


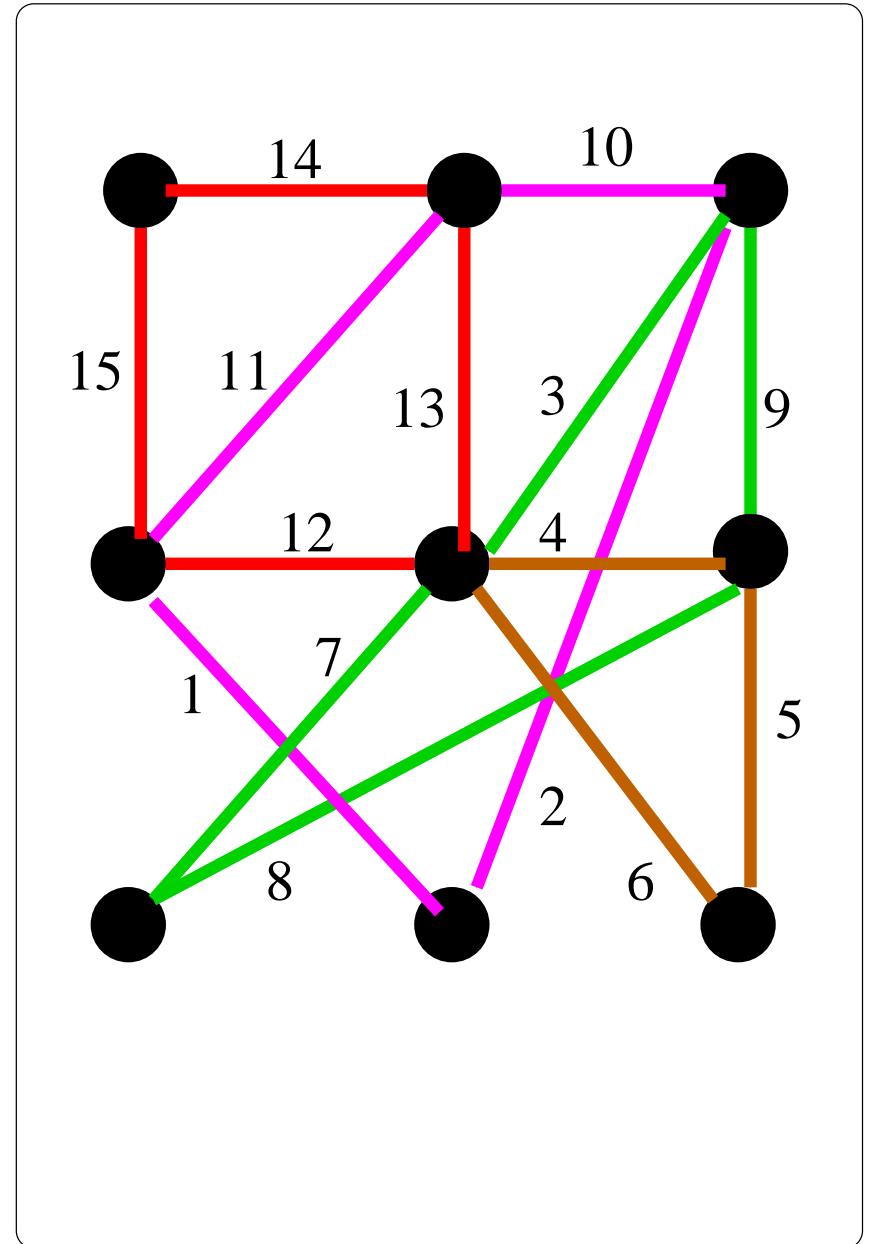
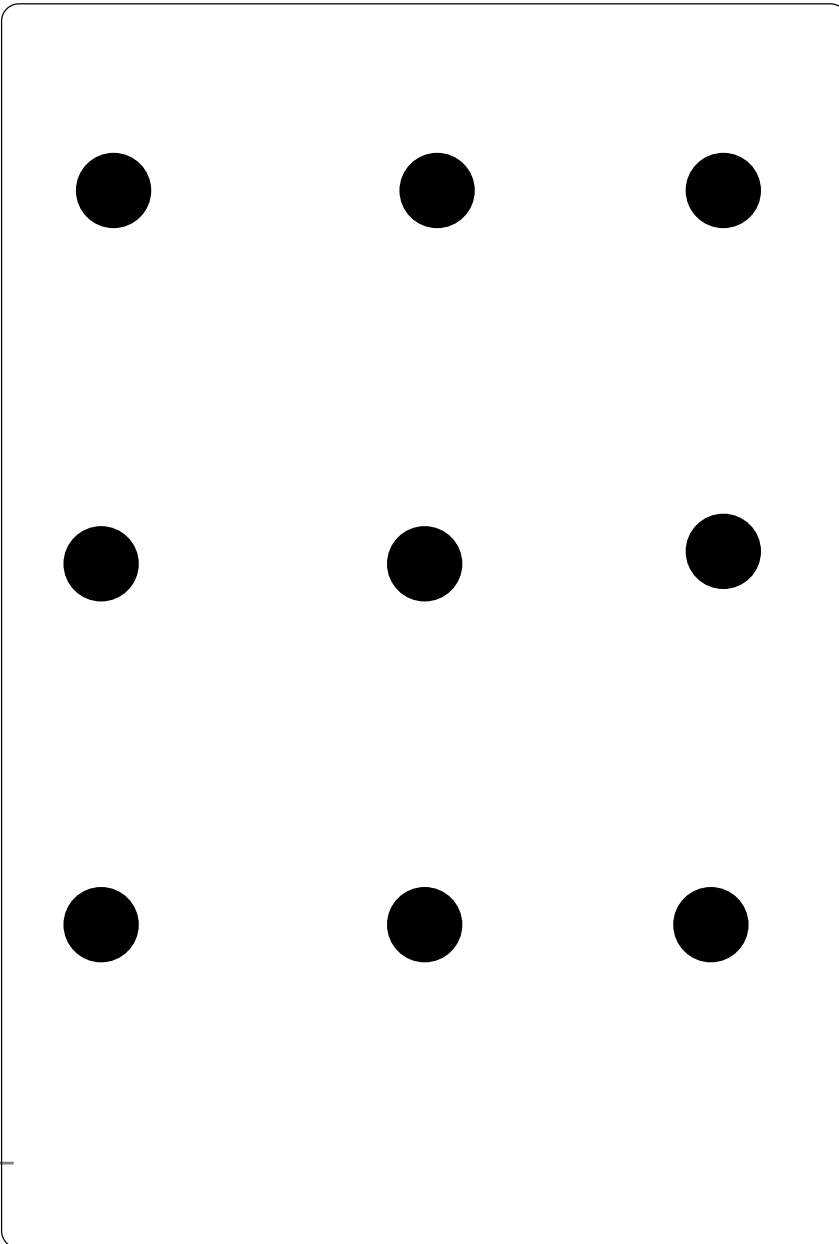












Algorithm 0.0.1: HIERHOLZER(G) $v \leftarrow$ dowolny element $V(G)$ $\mathcal{O} \leftarrow \text{CYKL}(G, v); G \leftarrow G - E(\mathcal{O})$ **while** $|E(G)| > 0$

do	{	$w \leftarrow 0$
		for each $v \in V(\mathcal{O})$
		do if $\Gamma_G(v) \neq \emptyset$ then $w \leftarrow v$; break
		if $w = 0$ then output („Graf nie jest eulerowski”); STOP
		$\mathcal{C} \leftarrow \text{CYKL}(G, w)$
		$\mathcal{O} \leftarrow \mathcal{O} + \mathcal{C}; G \leftarrow G - E(\mathcal{C})$
return		(\mathcal{O})

Zwróćmy uwagę, że operacja $G - E(\mathcal{O})$ odejmuje od grafu G tylko krawędzie szlaku domkniętego \mathcal{O} , natomiast operacja $\mathcal{O} + \mathcal{C}$ łączy w jeden szlak domknięty dwa szlaki domknięte mające wspólny wierzchołek w .

Algorithm 0.0.1: CYKL(G, v)**global** $Numer, Drzewo, Pozostale$ $Numer[x] \leftarrow Ponumerowano \leftarrow 1$ $NStos \leftarrow 1; Stos[NStos] \leftarrow x$ **while** $NStos > 0$

do {	{	$v \leftarrow Stos[NStos]$
		if $NI_v = 0$
		then $NStos \leftarrow NStos - 1$
		else {
		$w \leftarrow I_v[1]$ $NI_v \leftarrow NI_v - 1$ for $i \leftarrow 1$ to NI_v do $I_v[i] \leftarrow I_v[i - 1]$ if $Numer[w] = 0$ { $Ponumerowano \leftarrow Ponumerowano + 1$ $Numer[w] \leftarrow Ponumerowano$ $Drzewo \leftarrow Drzewo \cup \{vw\}$ $NStos \leftarrow NStos + 1; Stos[NStos] \leftarrow w$ }
		else $Pozostale \leftarrow Pozostale \cup \{vw\}$

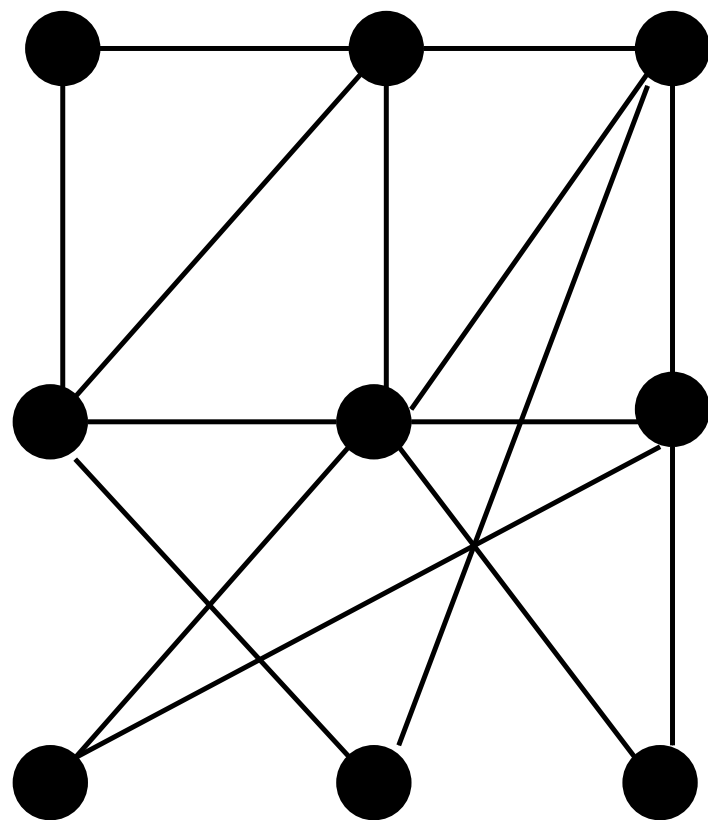
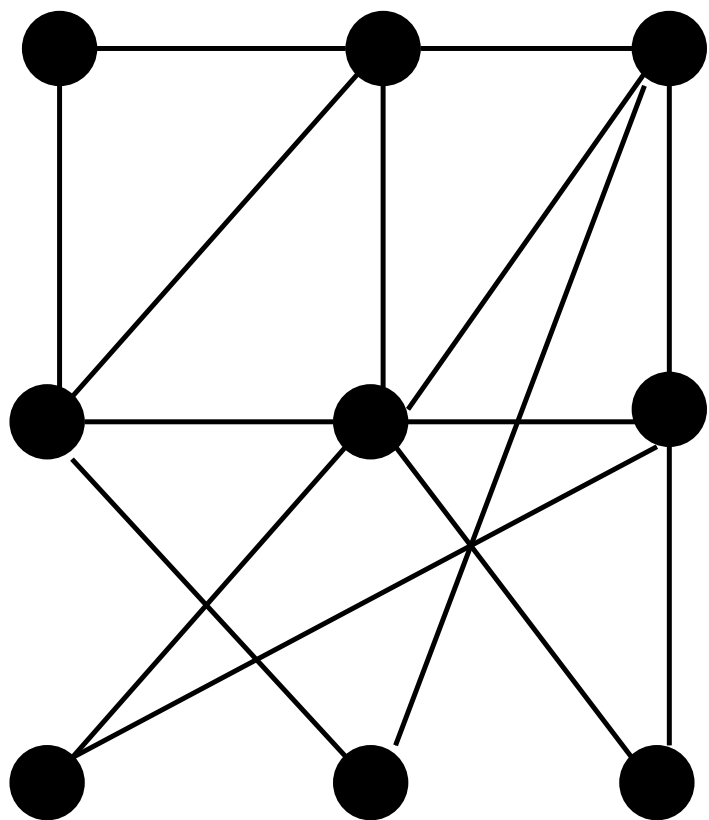
return ($Numer, Drzewo, Pozostale$)

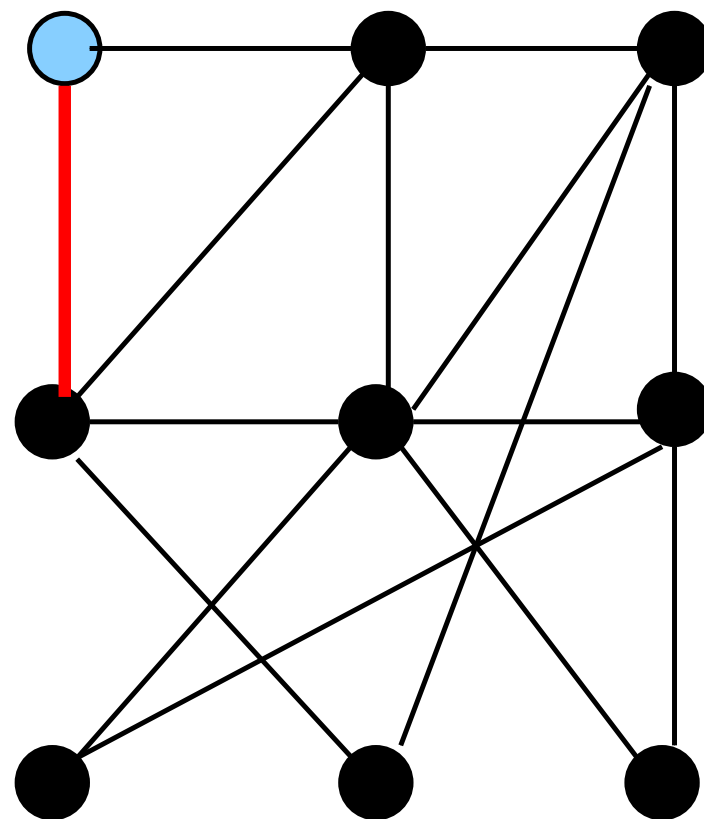
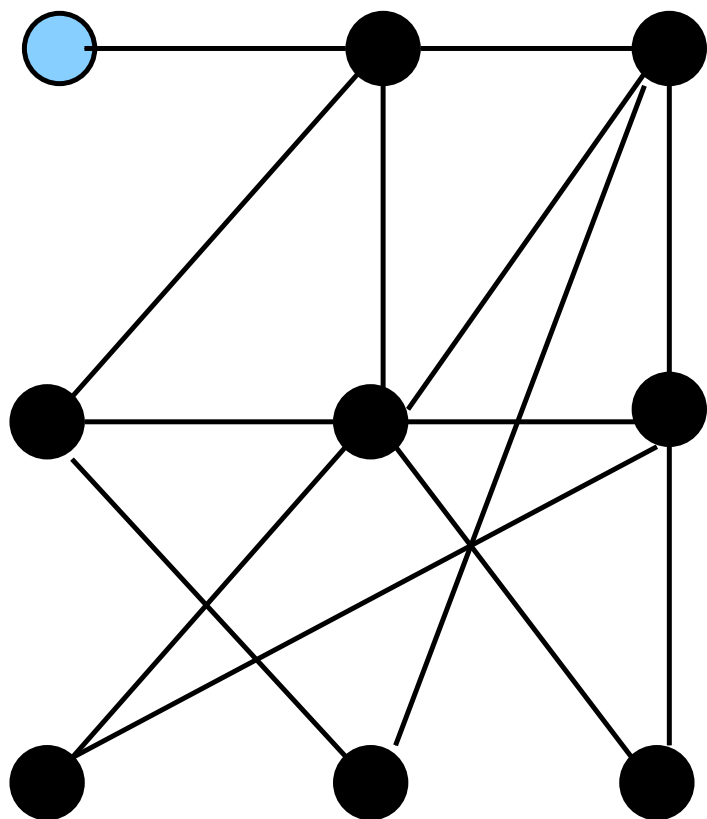
Do znalezienia cyklu w grafie G rozpoczynającego się i kończącego w wierzchołku $v \in V(G)$ możemy użyć nieco zmodyfikowanej procedury przeszukiwania grafu w głąb.

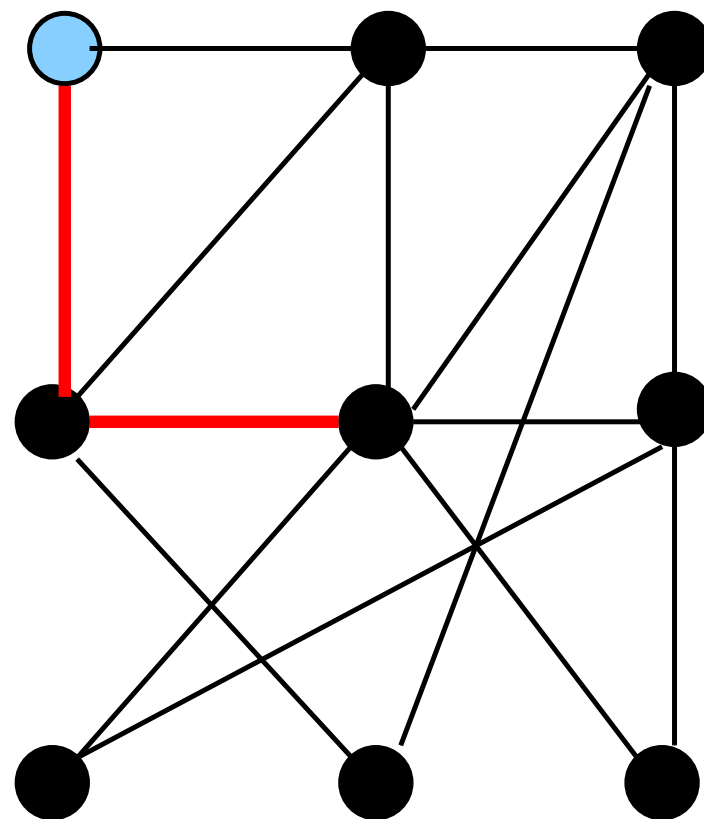
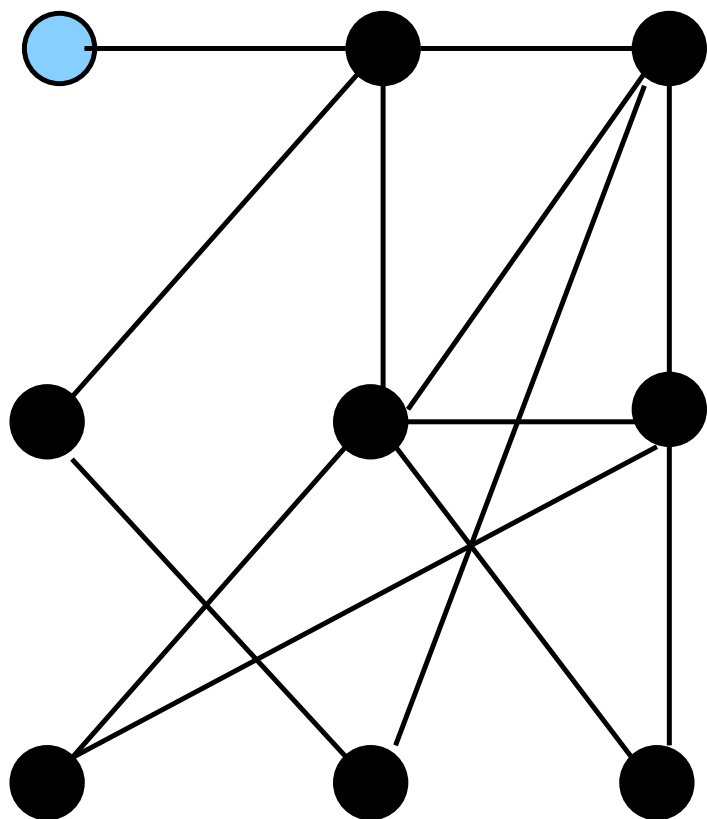
Algorytm Fleury'ego

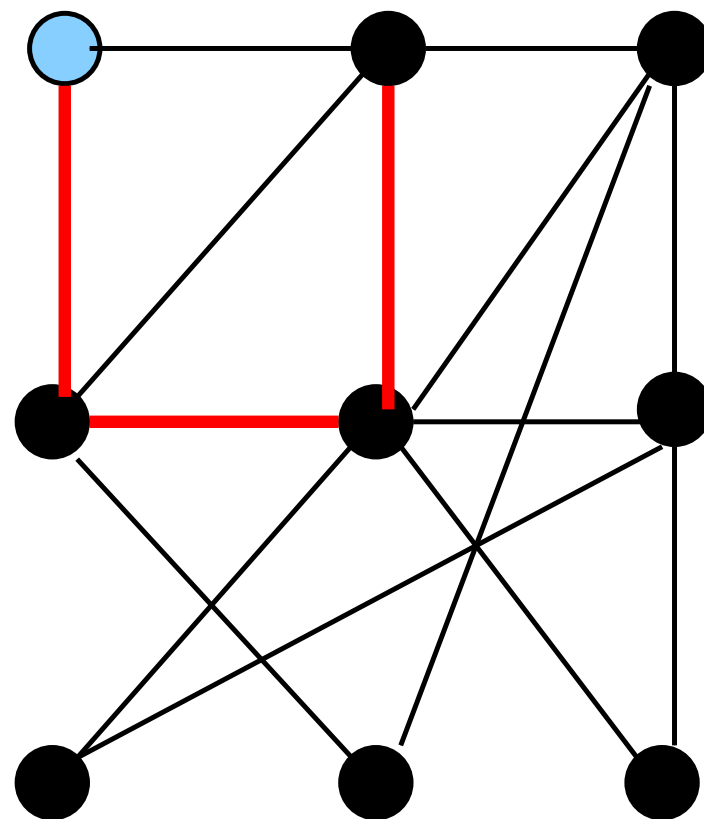
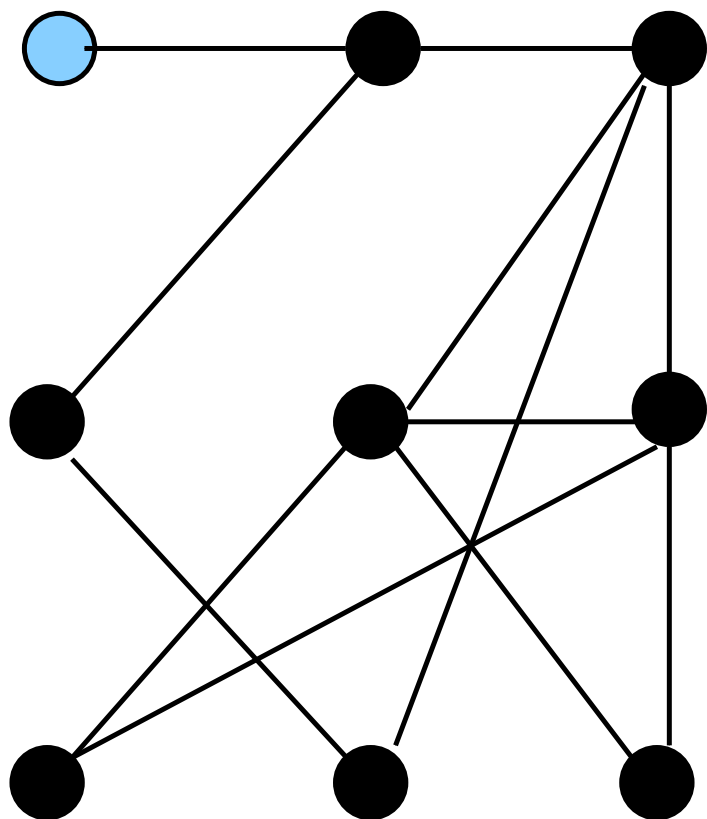
Algorytm Fleury'ego z 1883 roku polega na iteracyjnym rozszerzaniu istniejącego szlaku o kolejną krawędź, która, o ile jest to możliwe, nie jest mostem.

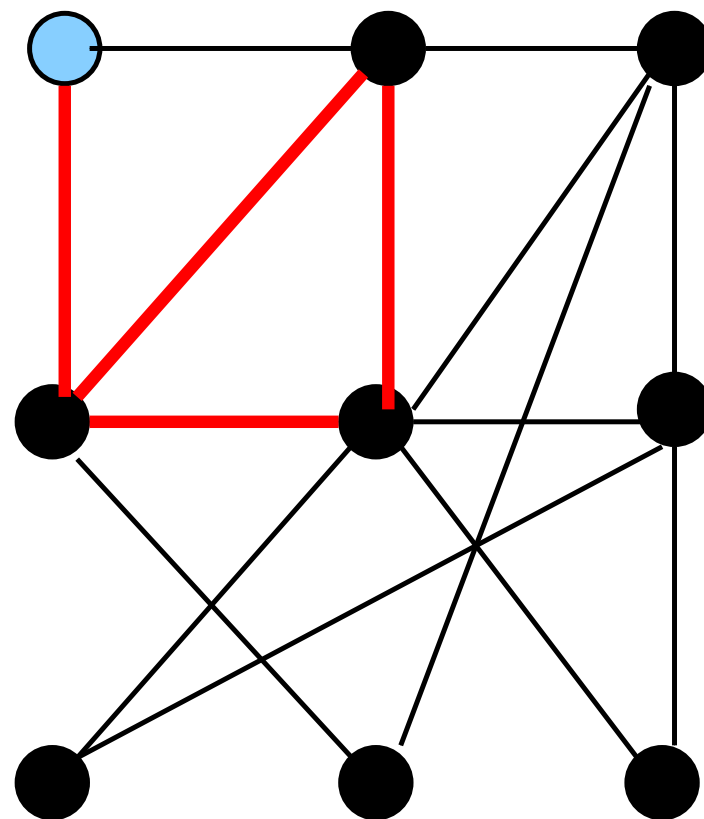
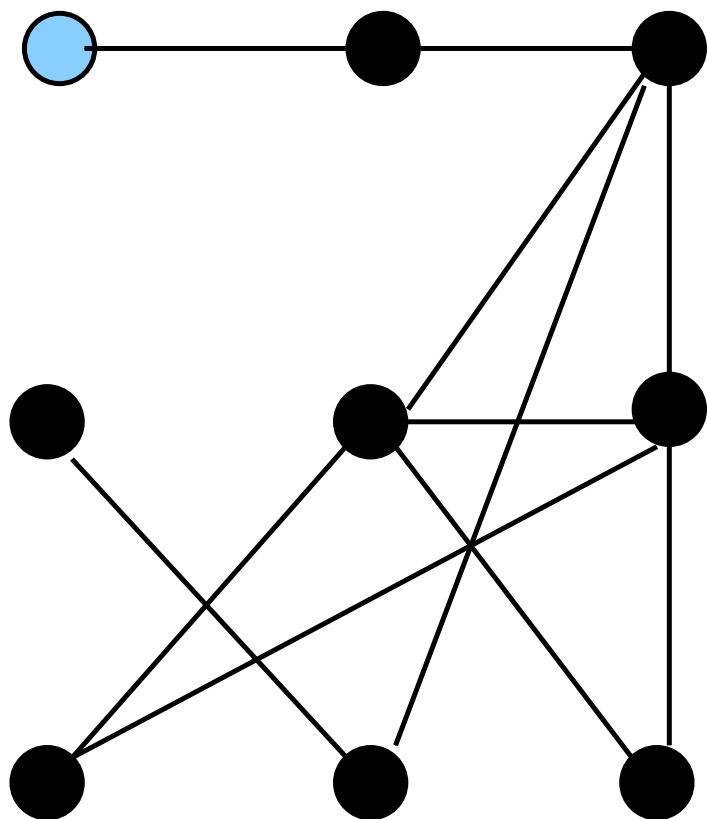
1. Wybierz dowolny wierzchołek v_0 i podstaw $W_0 = v_0$
2. Załóżmy, że szlak $W_i = v_0 e_1 v_1 \dots e_i v_i$ został wybrany.
Wybierz krawędź e_{i+1} z $E - \{e_1, e_2, \dots, e_i\}$ tak, by
 - (i) e_{i+1} była incydentna z v_i
 - (ii) e_{i+1} nie była krawędzią cięcia grafu $G_i = G - \{e_1, \dots, e_i\}$ chyba, że nie ma innej alternatywy
3. STOP jeżeli krok 2 nie może być wykonany.

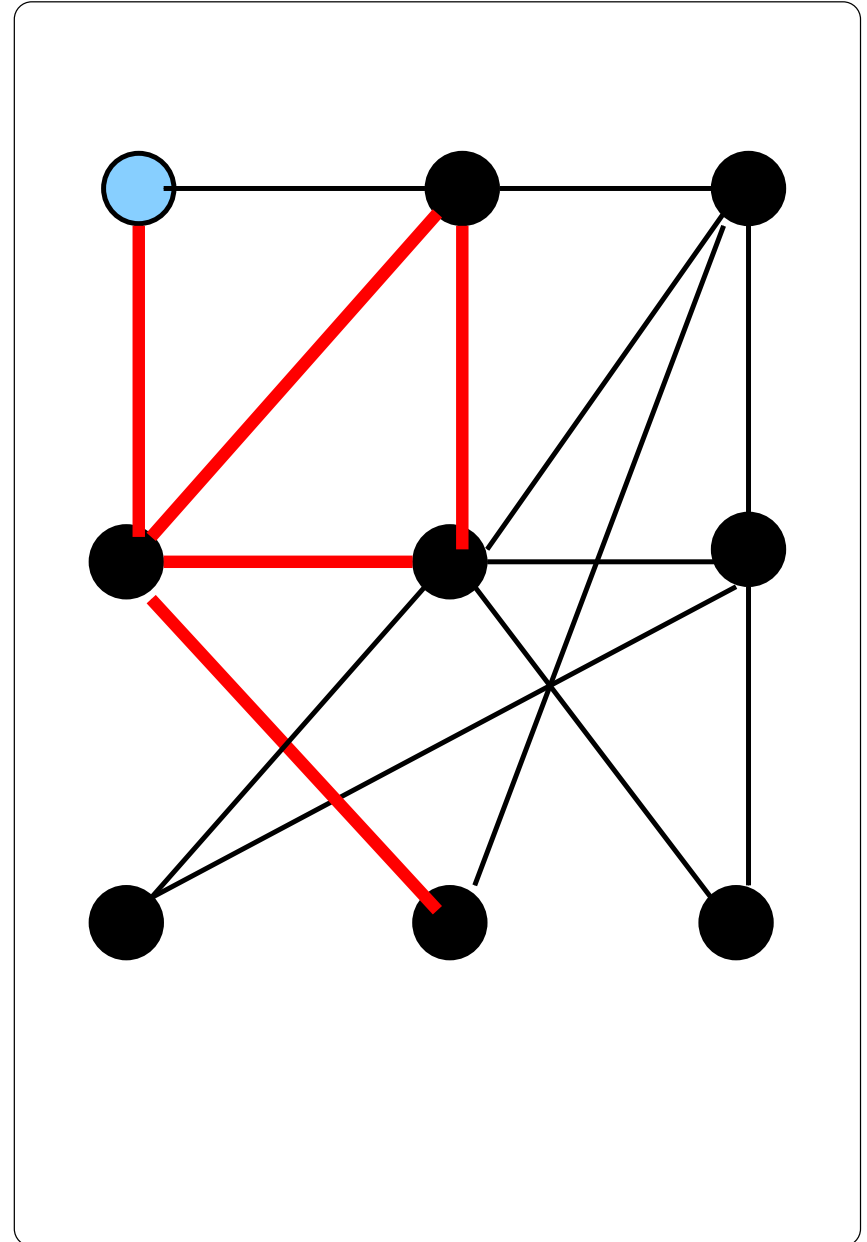
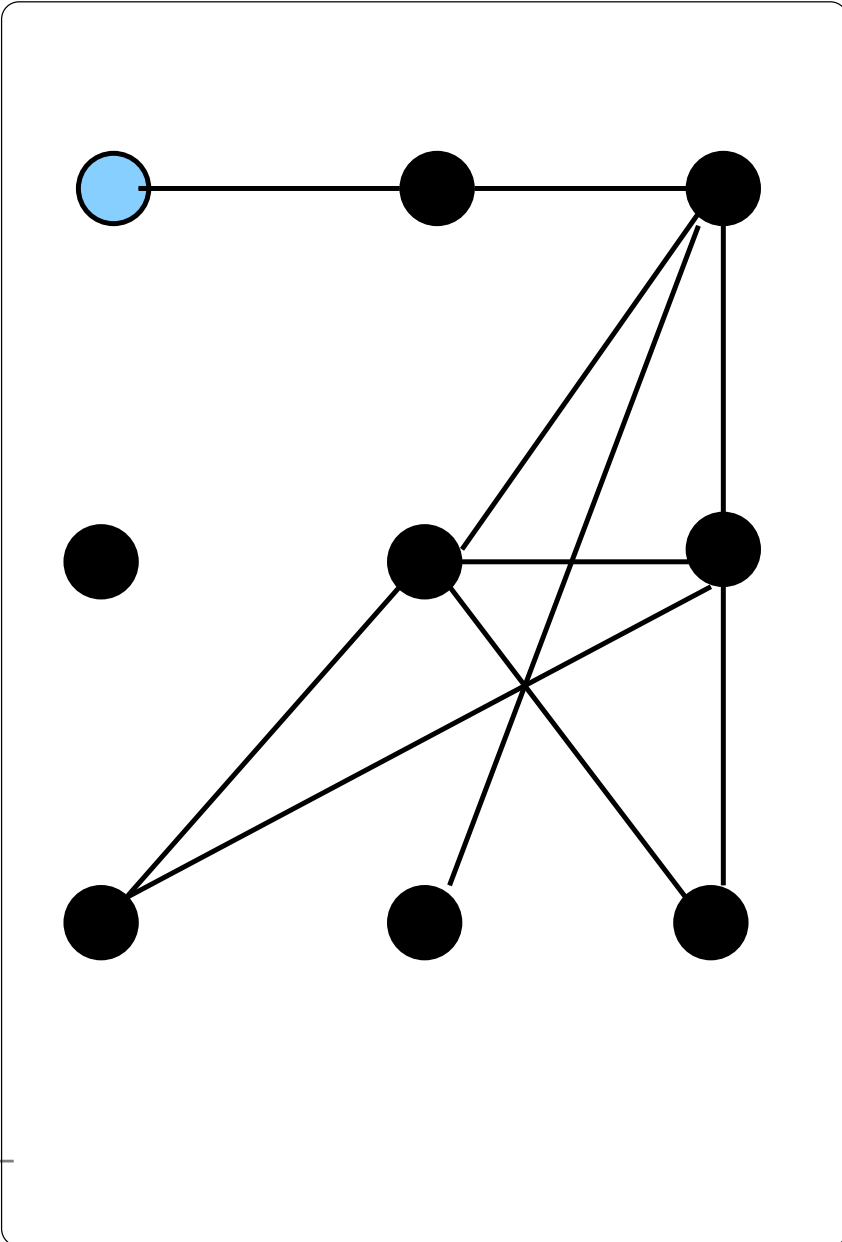


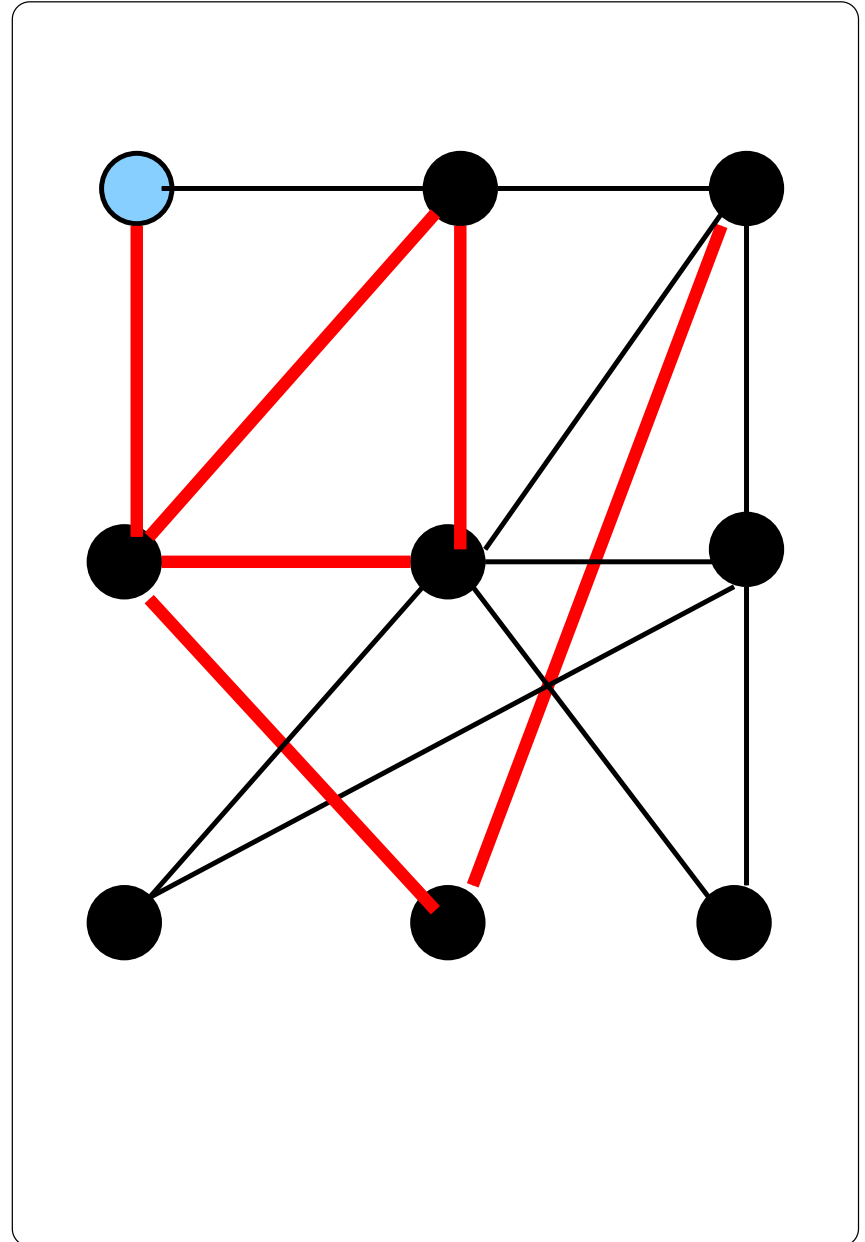
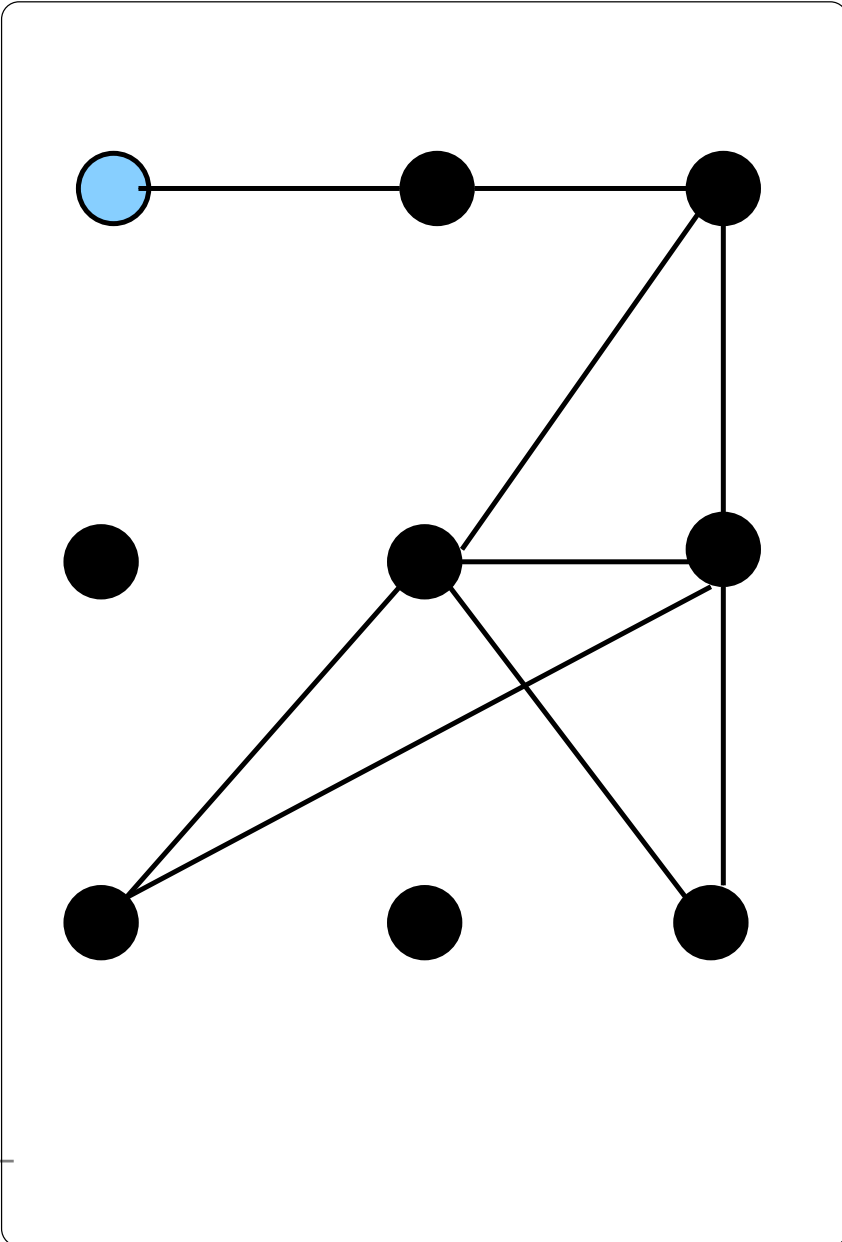


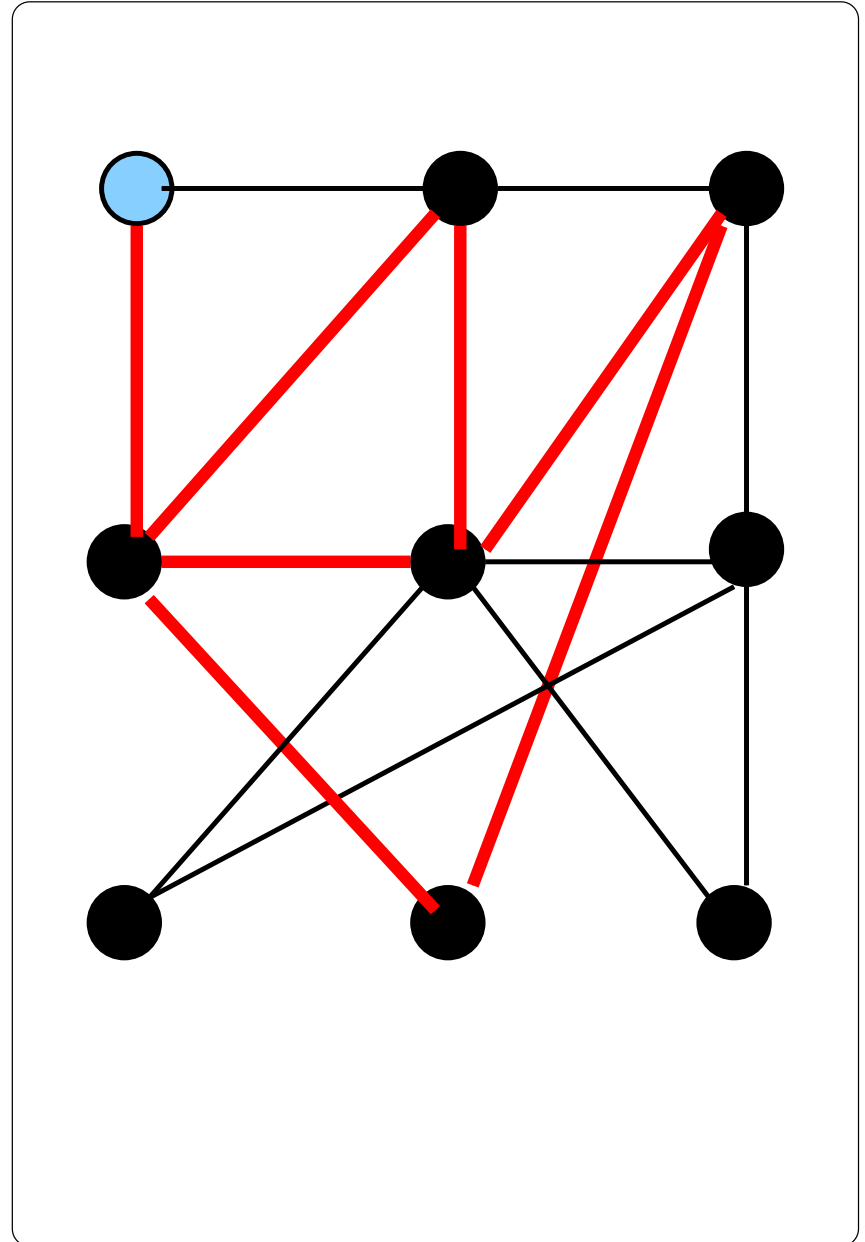
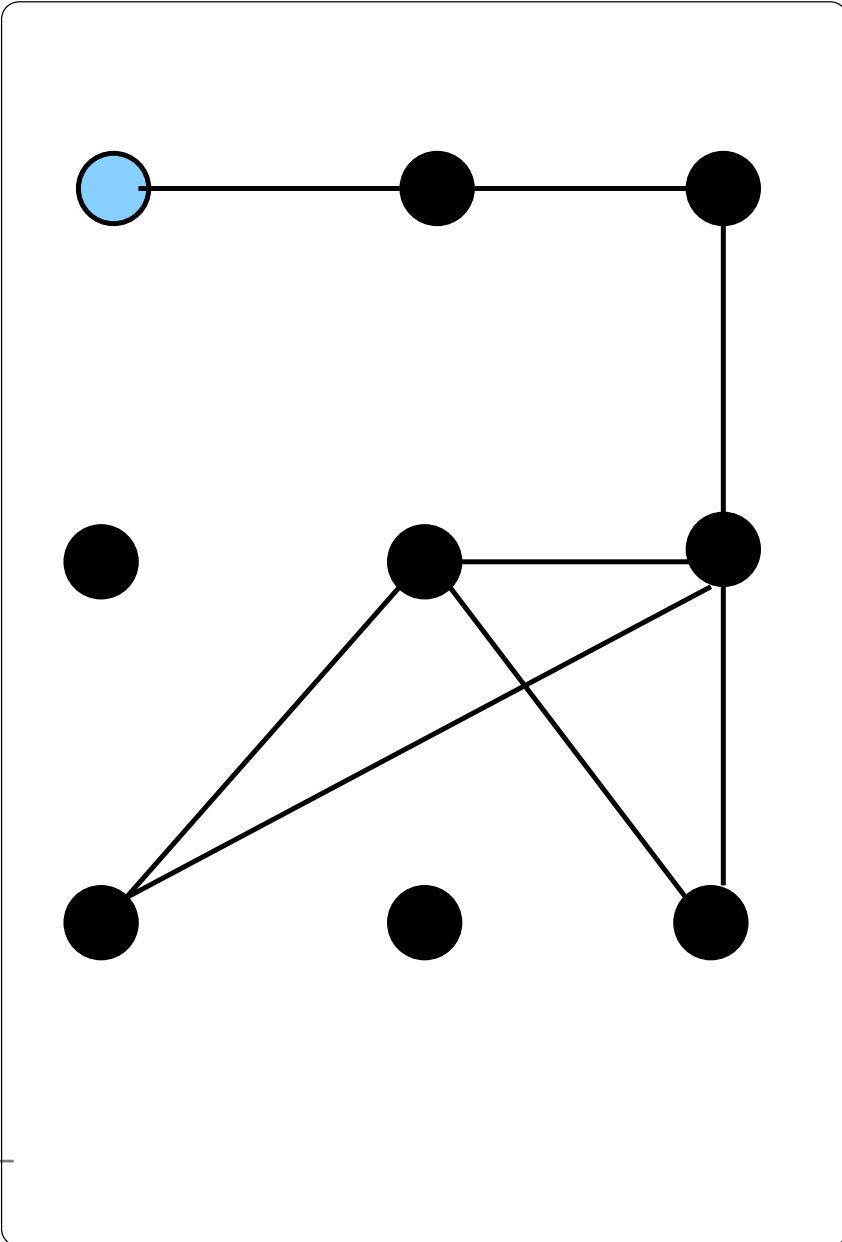


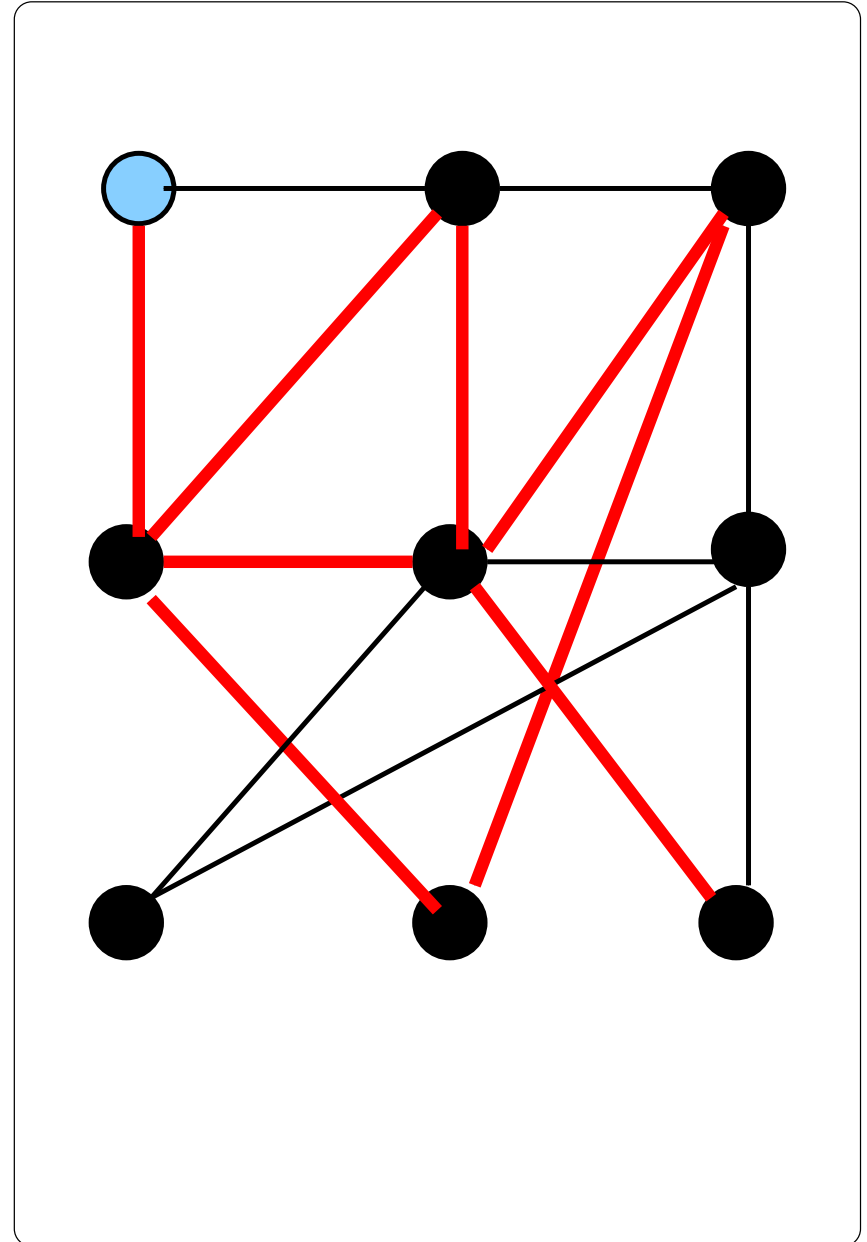
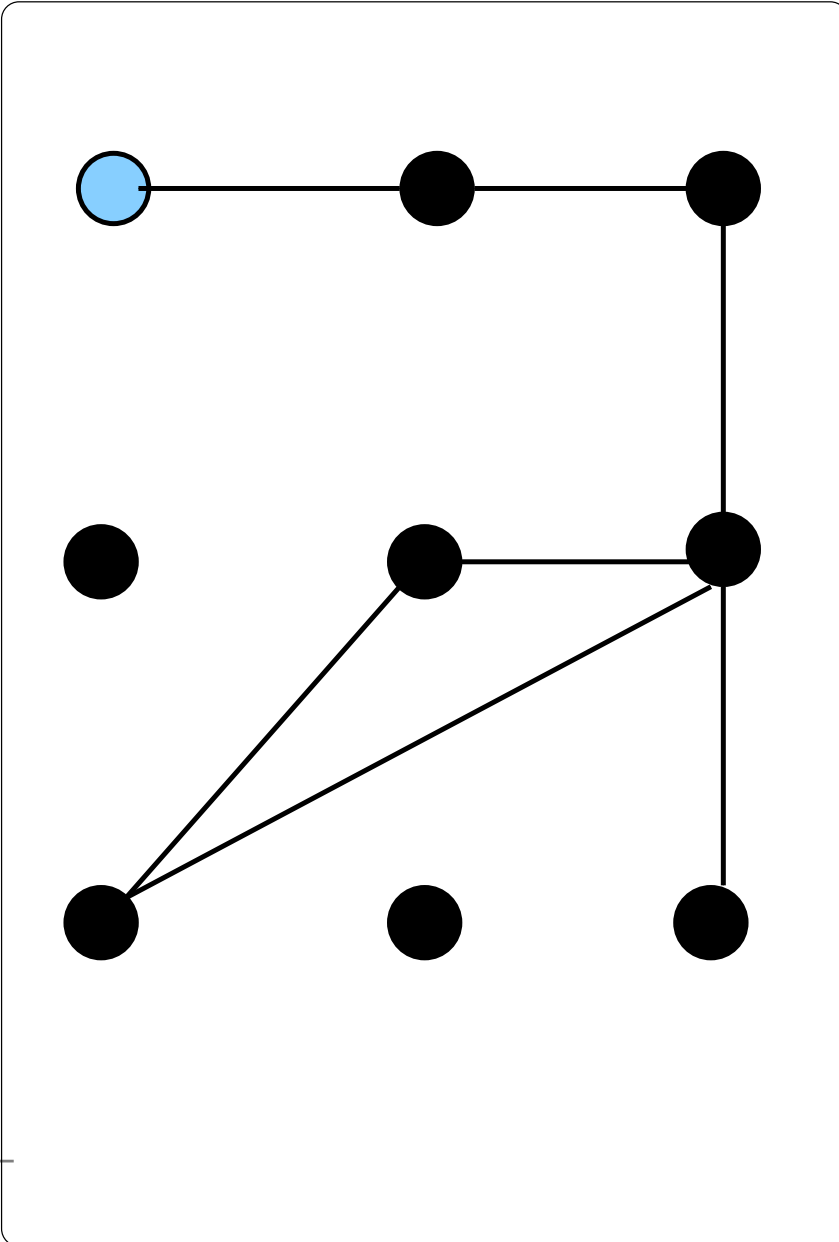


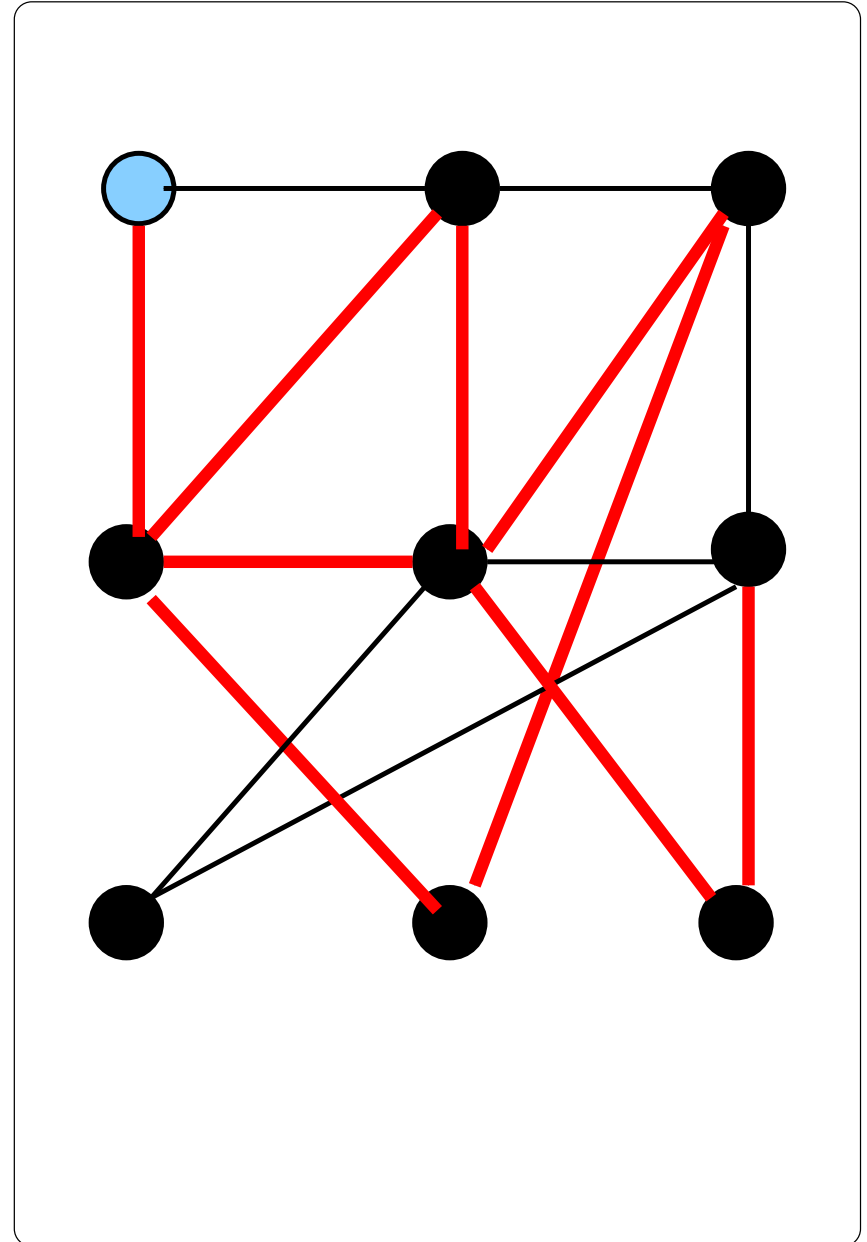
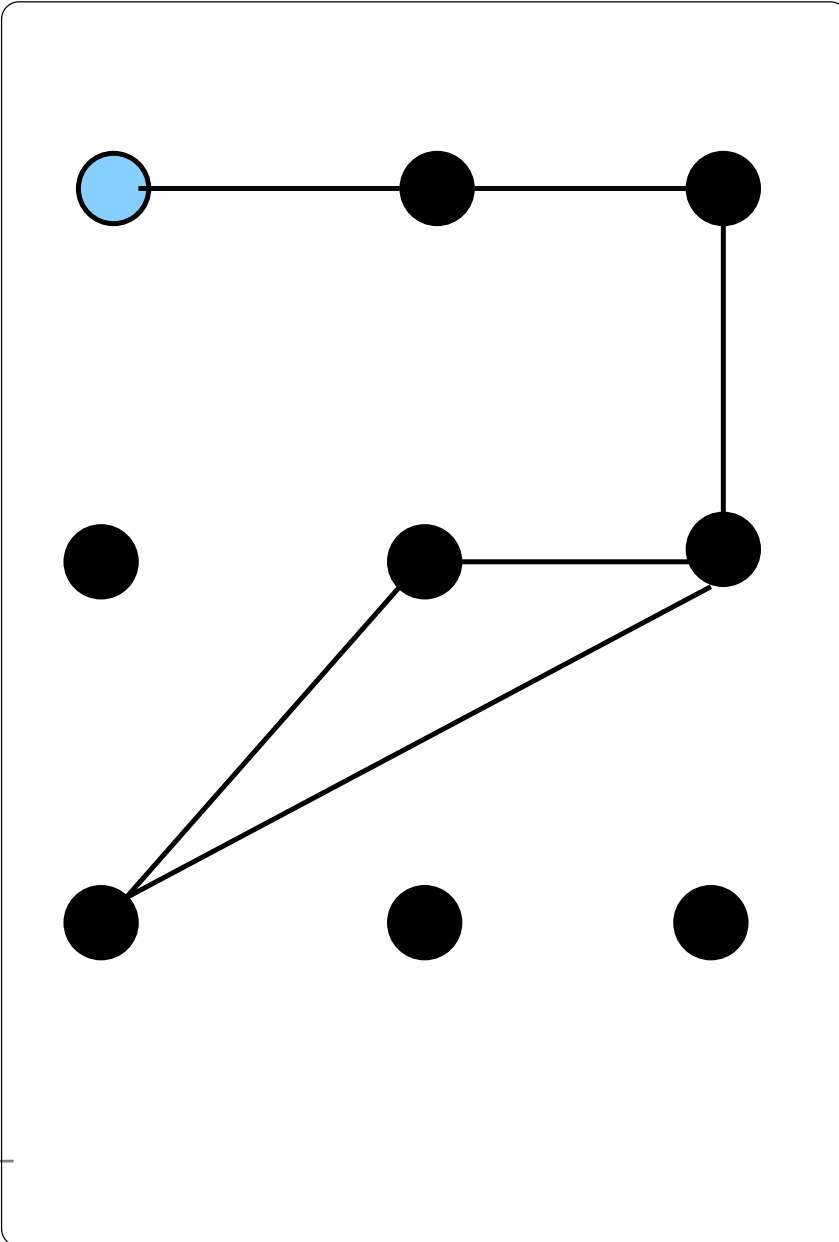


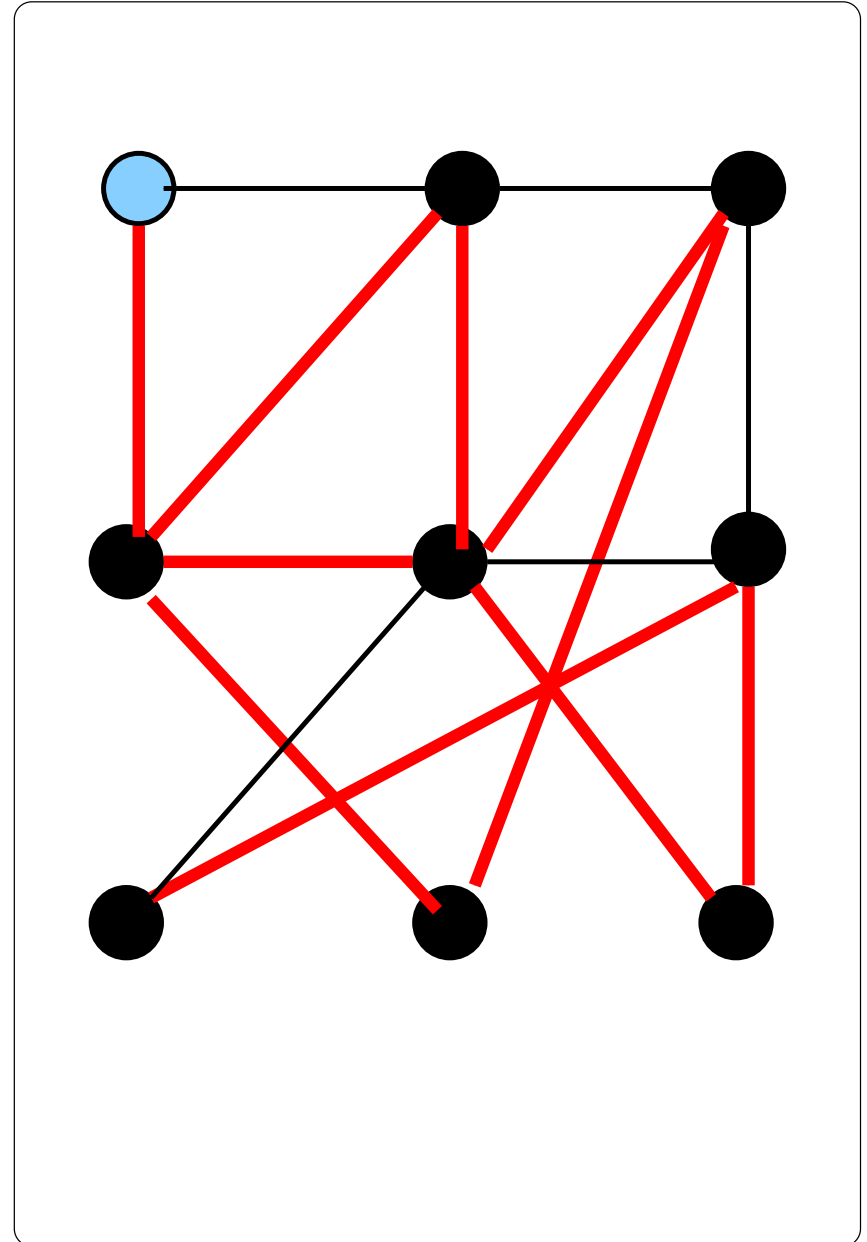
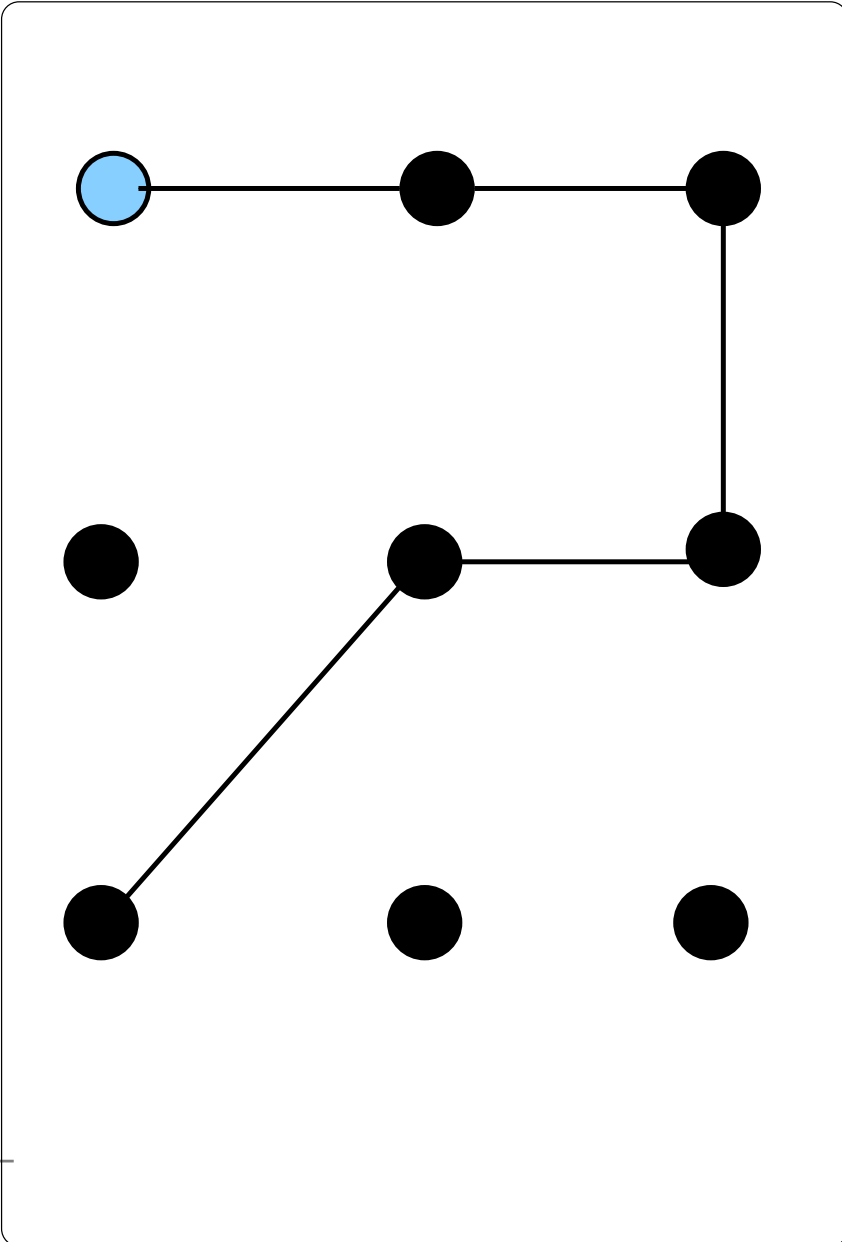


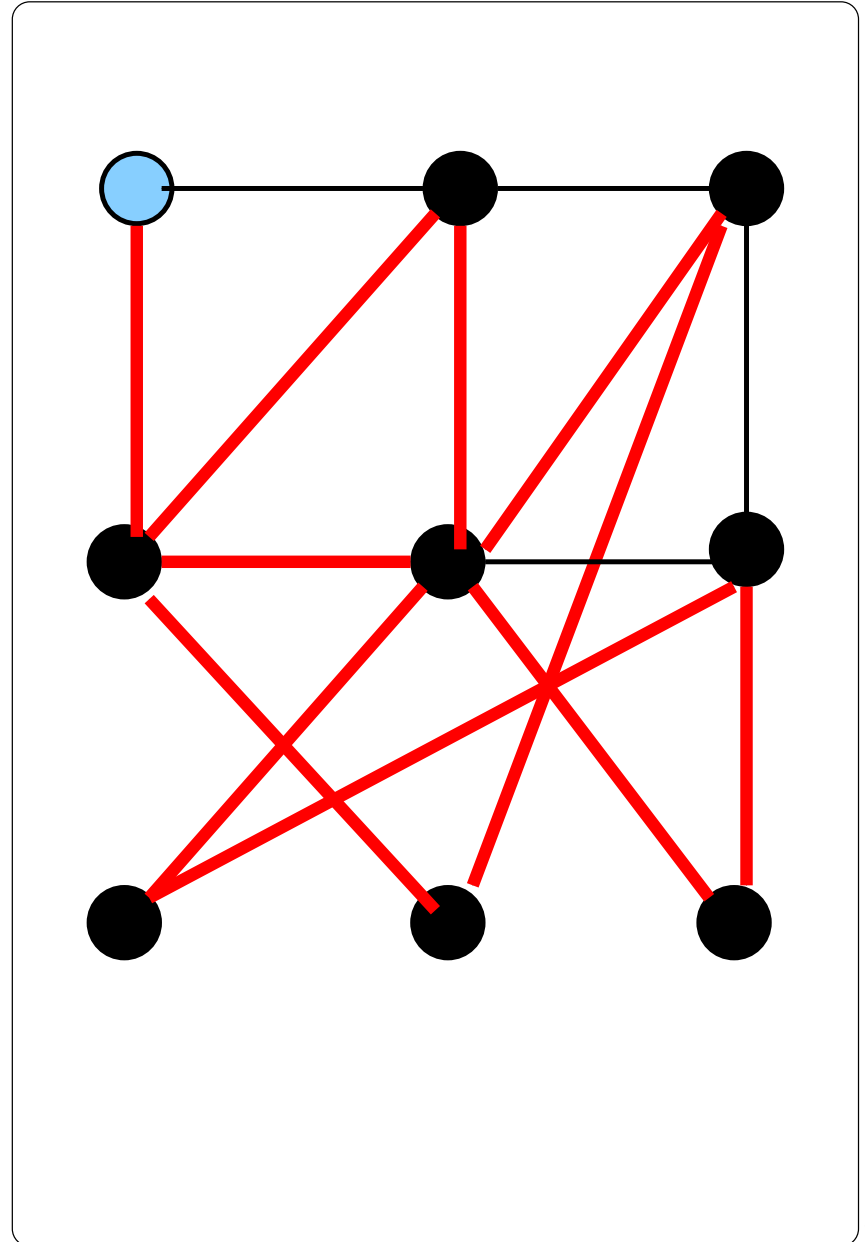
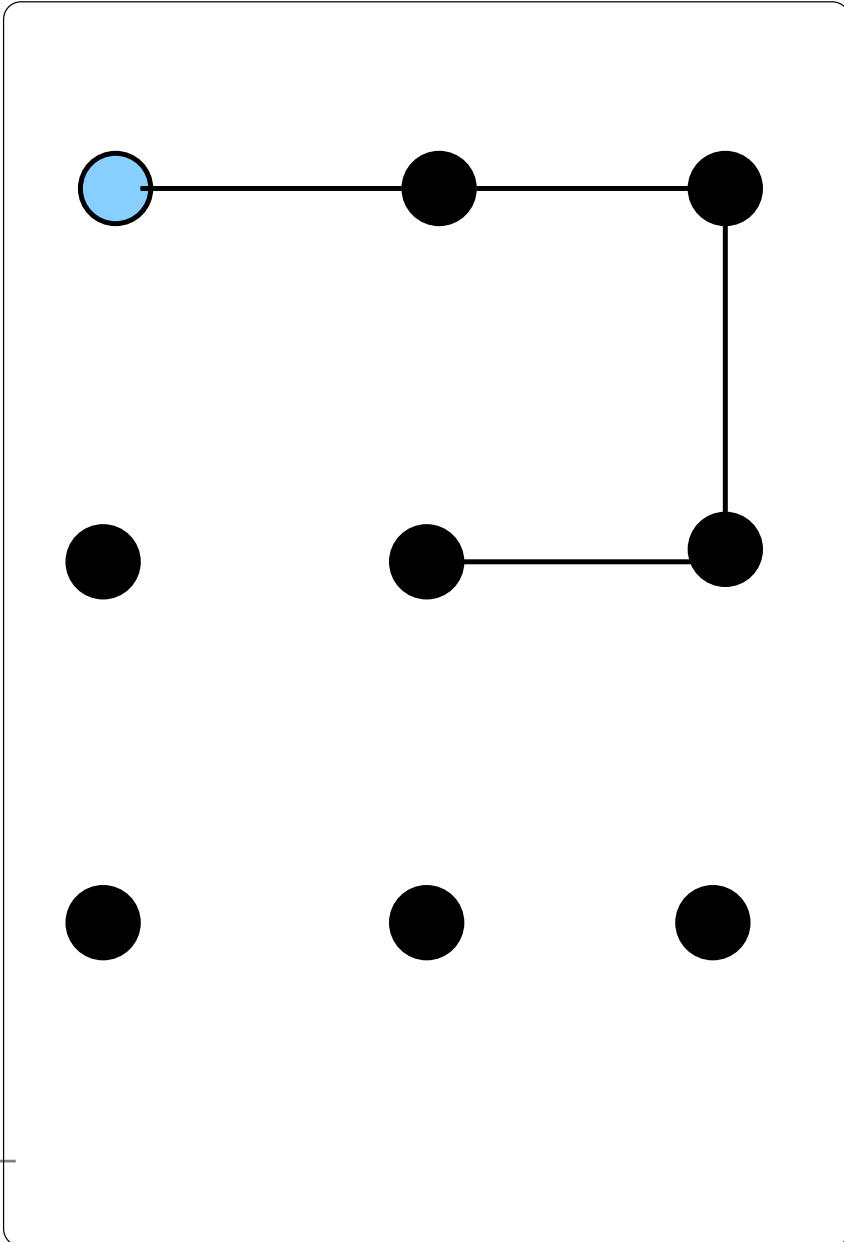


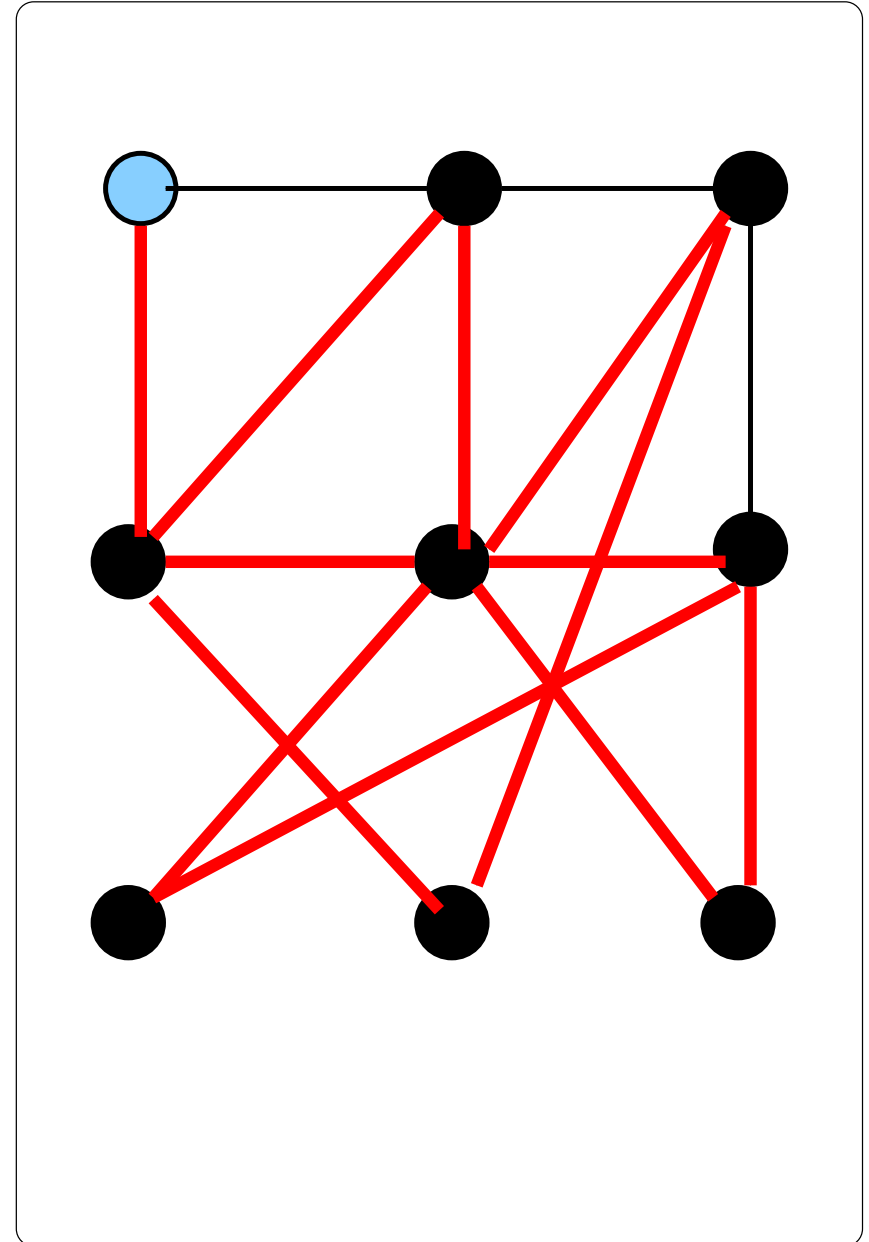
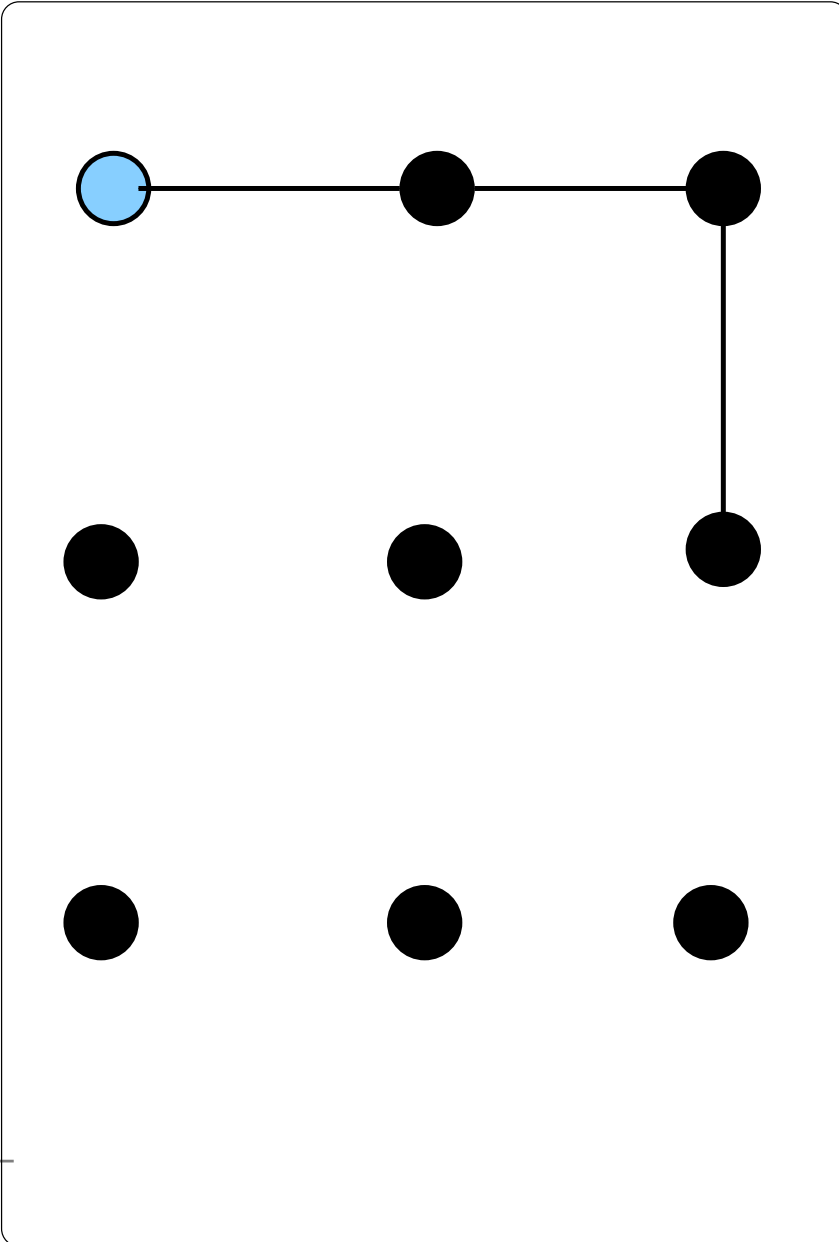


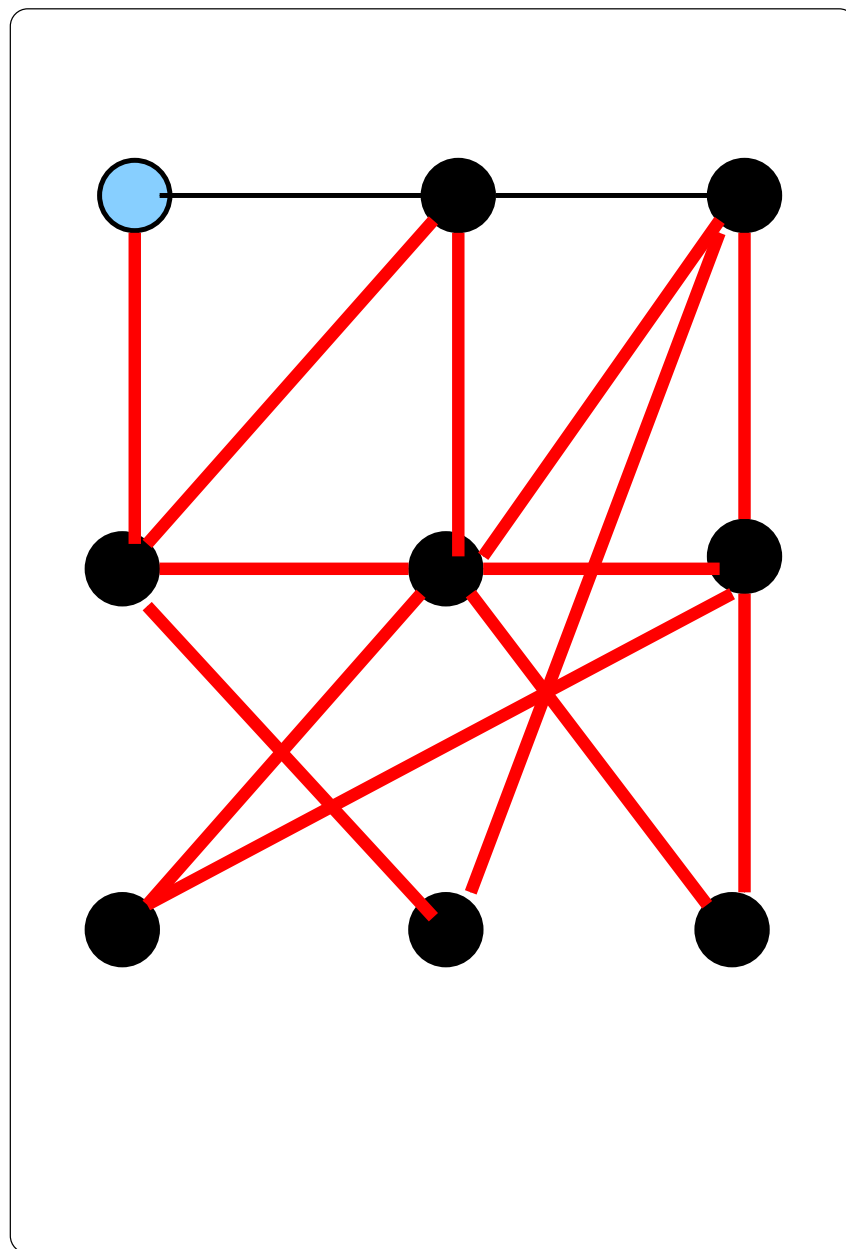
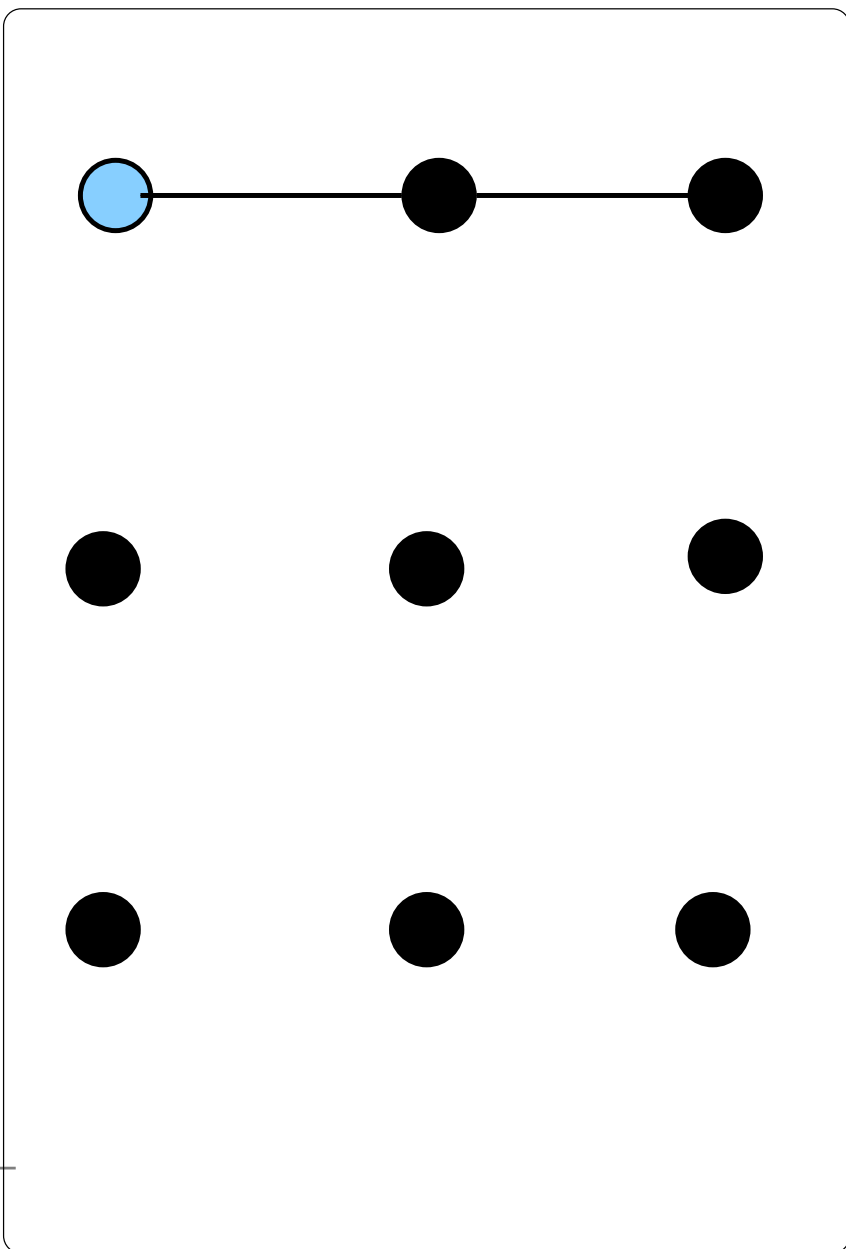


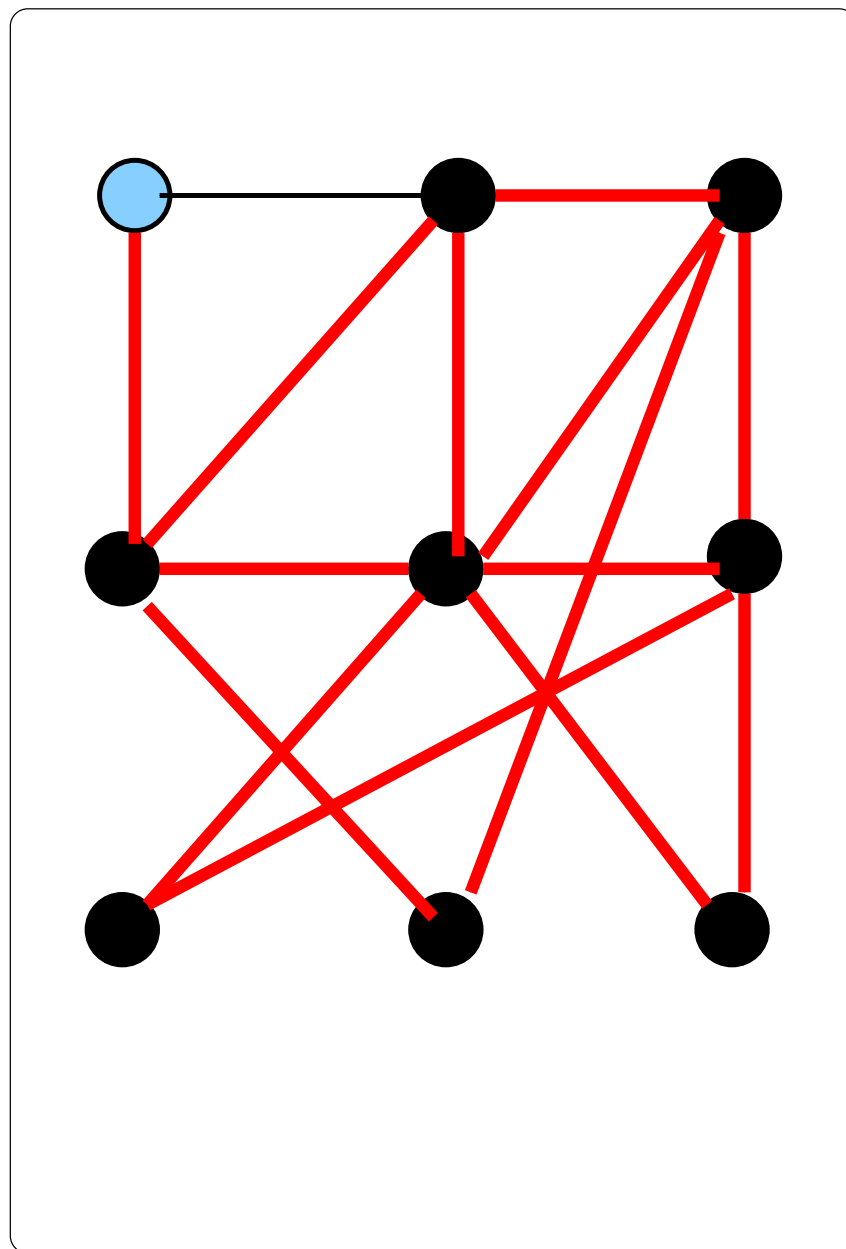
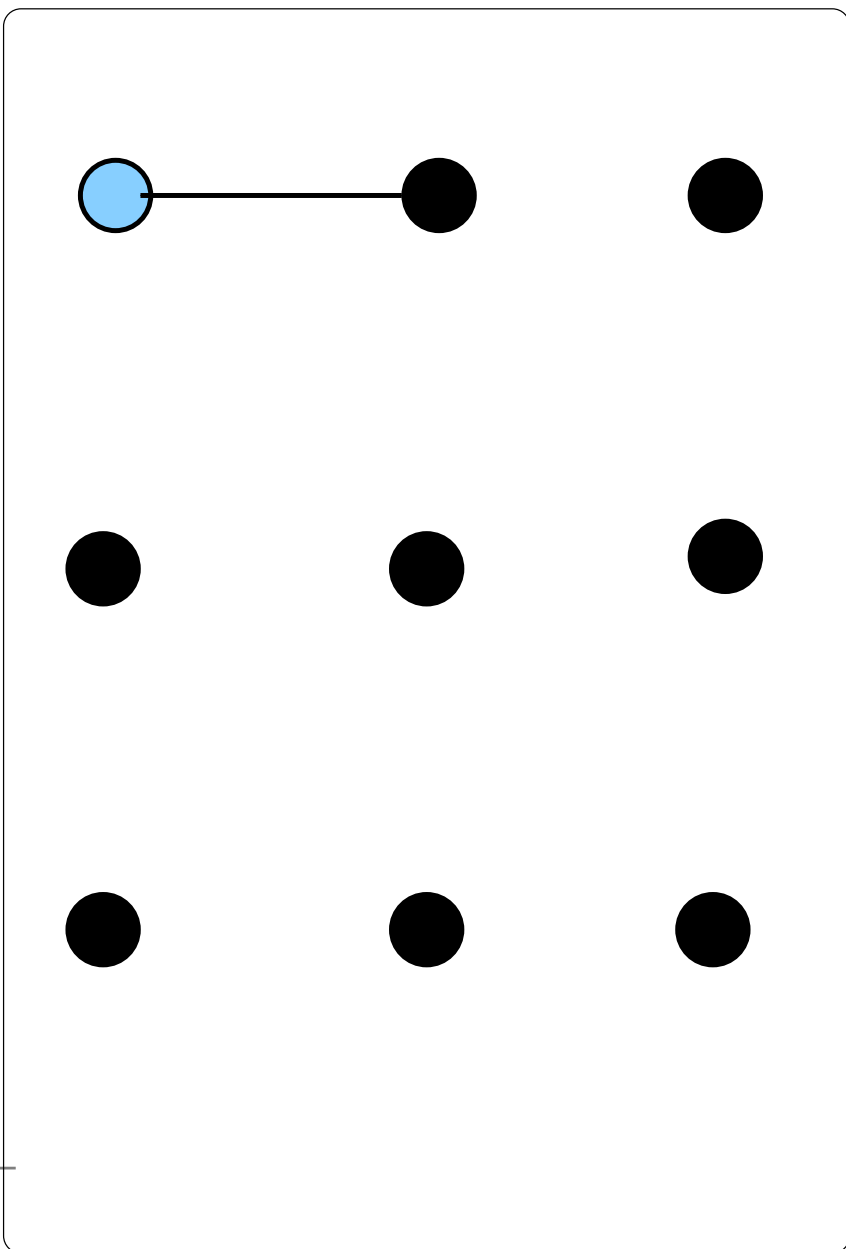


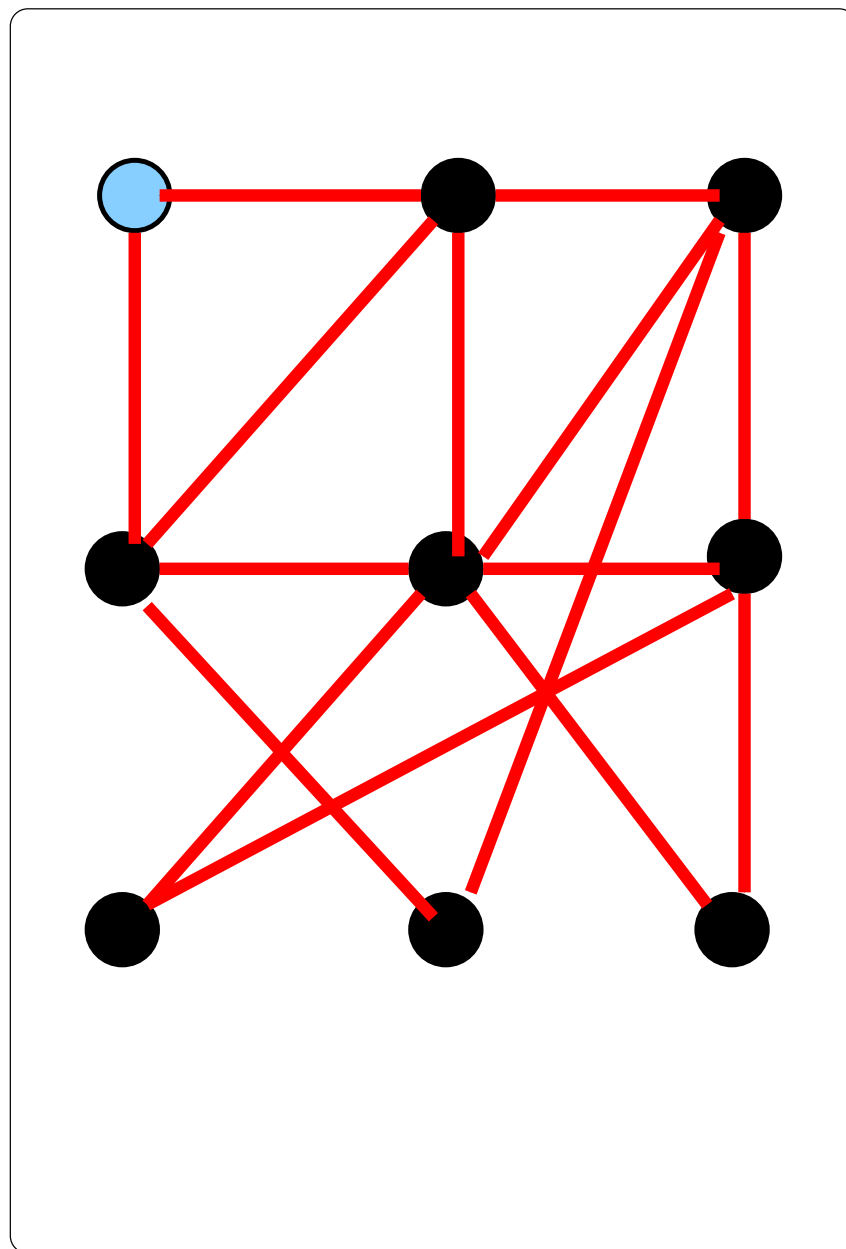
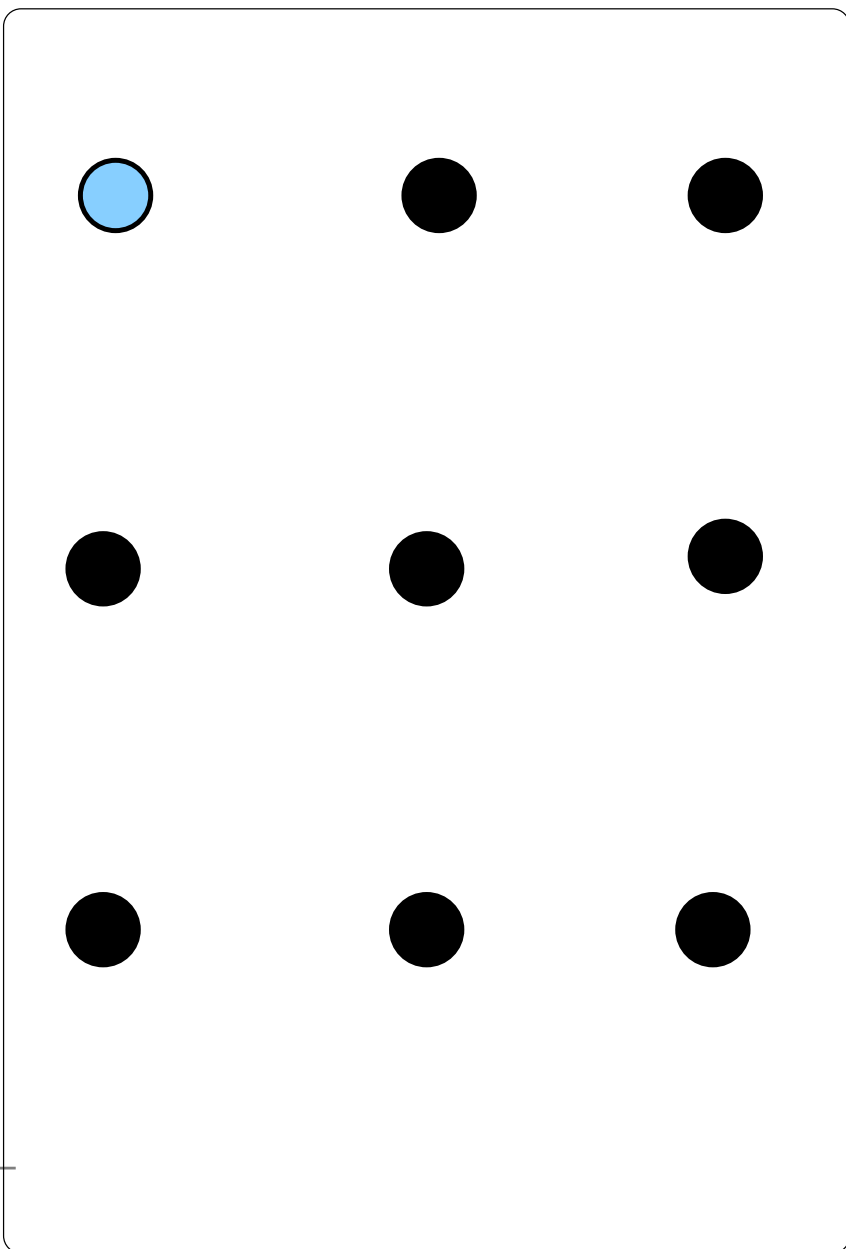












Algorithm 0.0.1: FLEURY(G)

```
 $v \leftarrow$  dowolny element  $V(G)$   
 $W \leftarrow v$   
while  $|E(G)| > 0$   
    if  $|\Gamma_G(v)| = 0$  then output („Graf nie jest eulerowski”); STOP  
     $w \leftarrow 0$   
    for each  $x \in \Gamma_G(v)$   
        do if  $\text{DISTBFS}(G - vx, v, x) < \infty$  then  $w \leftarrow x$ ; break  
    do  $\left\{ \begin{array}{l} \text{if } w = 0 \text{ then } w \leftarrow \text{dowolny element } \Gamma_G(v) \\ e \leftarrow \text{dowolna kraweć } vw \\ W \leftarrow W + e + w; G \leftarrow G - e \\ v \leftarrow w \end{array} \right.$   
return ( $W$ )
```

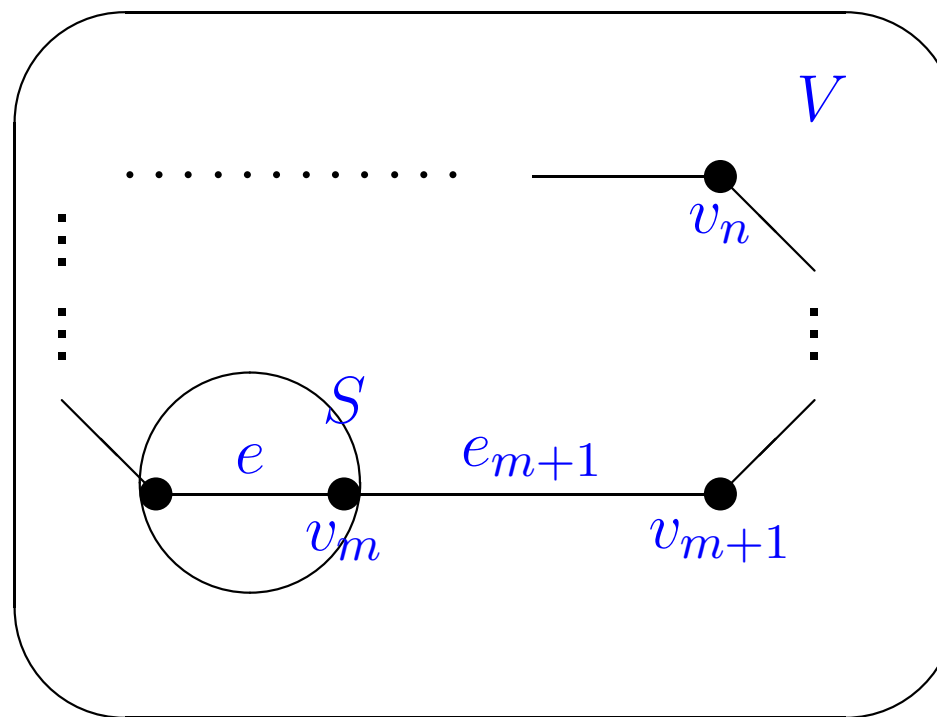
Algorytm ten jest sformalizowany w postaci pseudokodu FLEURY. Dla uproszenia zapisu i pominięcia indeksów w poniższym algorytmie v jest zawsze aktualnie ostatnim wierzchołkiem szlaku. Do sprawdzania, czy kraweć jest krawędzią cięcia użyliśmy procedury DISTBFS (patrz Część 3).

Twierdzenie . Jeżeli G jest grafem eulerowskim to dowolny szlak w G skonstruowany przy pomocy algorytmu Fleury'ego jest obchodem Eulera tego grafu.

Dowód.

- Niech G będzie grafem eulerowskim i niech $W_n = v_0 e_1 v_1 \dots e_n v_n$ będzie szlakiem w G otrzymanym przy pomocy algorytmu Fleury'ego. Ponieważ v_n ma stopień zero w $G_n = G - E(W_n)$ i wszystkie wierzchołki mają parzysty stopień w G , co oznacza, że $v_n = v_0$ czyli W_n jest szlakiem domkniętym.
- Przypuśćmy teraz, że W_n nie jest obchodem Eulera w G i niech S będzie zbiorem wierzchołków stopnia większego od zera w G_n . Zbiór S jest niepusty (z definicji), $S \cap V(W_n) \neq \emptyset$ (bo graf jest eulerowski) oraz $v_n = v_0 \in S^c$, gdzie $S^c = V \setminus S$, (bo nie można już przedłużyć szlaku).

- Niech m będzie największą liczbą całkowitą taką, że $v_m \in S$ a $v_{m+1} \in S^c$. Ponieważ W_n kończy się w S^c zatem e_{m+1} jest jedyną krawędzią zbioru krawędzi $[S, S^c]$ należącą do G_m , czyli jest ona krawędzią cięcia G_m (patrz poniższy rysunek).



- Niech e będzie dowolną krawędzią, różną od e_{m+1} , incydentną z wierzchołkiem v_m w grafie G_n (taka krawędź istnieje z definicji S). Z algorytmu wynika, że e musi być również krawędzią cięcia grafu G_m , a co za tym idzie krawędzią cięcia grafu $G_m[S]$. Ponieważ $G_m[S] = G_n[S]$, to łatwo zauważyć, że każdy wierzchołek w $G_m[S]$ ma stopień parzysty co pociąga, że $G_m[S]$ nie może mieć krawędzi cięcia czyli doszliśmy do sprzeczności.



8.3 Problem Chińskiego Listonosza

- Listonosz odbiera przesyłki z poczty, dostarcza je adresatom a następnie wraca na pocztę. Zakładamy, że dla każdego adresata ma przesyłkę, więc musi przejść przez każdą ulicę w swoim rejonie przynajmniej raz. Ze względu na ten warunek pragnie wybrać obchód w taki sposób by jak najmniej „spacerować”. Problem ten znany jest jako *Problem Chińskiego Listonosza*, ponieważ pierwszy rozpatrywał go chiński matematyk Guan Meigu w 1962 roku.

- W grafie z wagami definiujemy *wagę* obchodu jako sumę wag jego krawędzi.
- *Problem Chińskiego Listonosza* sprowadza się do znalezienia obchodu o minimalnej wadze w spójnym grafie o nieujemnych wagach.
- Jeżeli G jest eulerowski, to obchód Eulera jest optymalny, ponieważ jest obchodem przechodzącym przez każdą krawędź dokładnie jeden raz.
- *Problem Chińskiego Listonosza* ma wówczas proste rozwiązanie, ponieważ istnieją efektywne algorytmy znajdowania obchodu Eulera w grafie eulerowskim, na przykład przedstawione tu algorytmy Hierholzera oraz Fleury'ego.

- W ogólnym przypadku, gdy graf nie jest eulerowski, listonosz musi przechodzić niektórymi ulicami dwukrotnie (wielokrotnie). W celu sprowadzenia przypadku ogólnego do problemu eulerowskiego wprowadźmy operację duplikowania krawędzi.

Definicja (Duplikacja krawędzi). *Duplikacja krawędzi e , o wadze $w(e)$, jest operacją dodania nowej krawędzi łączącej końce krawędzi e i mającej wagę $w(e)$ (powstaje krawędź wielokrotna).*

- Zauważmy, że wtedy możemy przeformułować problem chińskiego listonosza w następujący sposób:

Dany jest graf G o nieujemnych wagach krawędzi:

- (i) znaleźć za pomocą duplikowania krawędzi eulerowski ważony nadgraf G^* grafu G taki, że

$$\sum_{e \in E(G^*) - E(G)} w(e)$$

jest najmniejsza z możliwych,

- (ii) znaleźć obchód Eulera w grafie G^* .

• *Zauważmy, że część (ii) potrafimy już rozwiązać. Możemy to zrobić przy pomocy efektywnego algorytmu Fleury'ego lub Hierholzera. Efektywny (wielomianowy) algorytm rozwiązania części (i) podali Edmonds i Johnson w 1973 roku.*

- Niech V^- będzie zbiorem wierzchołków stopnia nieparzystego grafu G (oczywiście $|V^-|$ jest parzysta) oraz niech \mathcal{M} będzie zbiorem krawędzi dodanych do grafu G (zduplikowanych) w celu otrzymania grafu eulerowskiego. Zbiór \mathcal{M} składa się ze spacerów grafu G , które kojarzą w pary wierzchołki z V^- i ewentualnie z cykli.
- Oznaczmy przez graf $G^+(\mathcal{M})$ multigraf (graf z krawędziami wielokrotnymi) otrzymany z G poprzez duplikację krawędzi spacerów i cykli z \mathcal{M} , a dokładniej zastąpienie każdej krawędzi grafu G wiązką $k + 1$ krawędzi o takiej samej wadze, jeżeli krawędź ta została wykorzystana k razy w spacerach i cyklach z \mathcal{M} (0 jeżeli nie była ona wykorzystana).

Twierdzenie . Dla optymalnego obchodu grafu G istnieje zbiór ścieżek \mathcal{M} kojarzących wierzchołki stopnia nieparzystego w pary, taki że suma wag krawędzi tych ścieżek jest minimalna i obchód Eulera grafu $G^+(\mathcal{M})$ jest rozwiązaniem Problemu Chińskiego Listonosza.

Dowód. Jeżeli wyjściowy obchód optymalny przechodził przez krawędź l razy, to w grafie $G^+(\mathcal{M})$ w miejscu tej krawędzi istnieje wiązka l krawędzi.

Zwróćmy uwagę, że krawędzie zduplikowane (tj. należące do \mathcal{M}) nie zawierają żadnego cyklu. Gdyby zawierały cykl, to po jego usunięciu graf zawierałby obchód Eulera o mniejszej wadze.

Weźmy dowolny wierzchołek stopnia nieparzystego v_1 w grafie G .

Optymalny obchód \mathcal{O} musiał oczywiście wykorzystać jedną z krawędzi incydentnych do v_1 więcej niż jeden raz. Załóżmy, że jest to krawędź $e_1 = (v_1, v_2)$. Duplikujemy e_1 i przechodzimy do v_2 .

Jeżeli v_2 jest wierzchołkiem stopnia nieparzystego to $v_1 e_1 v_2$ jest pierwszą szukaną ścieżką kojarzącą wierzchołki stopnia nieparzystego v_1 i v_2 . Jeżeli v_2 był wierzchołkiem stopnia parzystego, to po duplikacji e_1 będzie on już nieparzystego stopnia i któraś z krawędzi do niego incydentnych musiała być wykorzystana przez obchód \mathcal{O} więcej niż jeden raz. Oczywiście znajdziemy w ten sposób ścieżki kojarzące w pary wierzchołki stopnia nieparzystego. Nietrudno również zauważyć, że nie ma więcej krawędzi zduplikowanych. Po dodaniu do grafu G wybranych już ścieżek wszystkie wierzchołki nowo powstałego grafu mają parzyste stopnie. Gdyby istniały jeszcze jakieś zduplikowane krawędzie, musiałyby tworzyć cykle, a to już wykluczyliśmy. ■

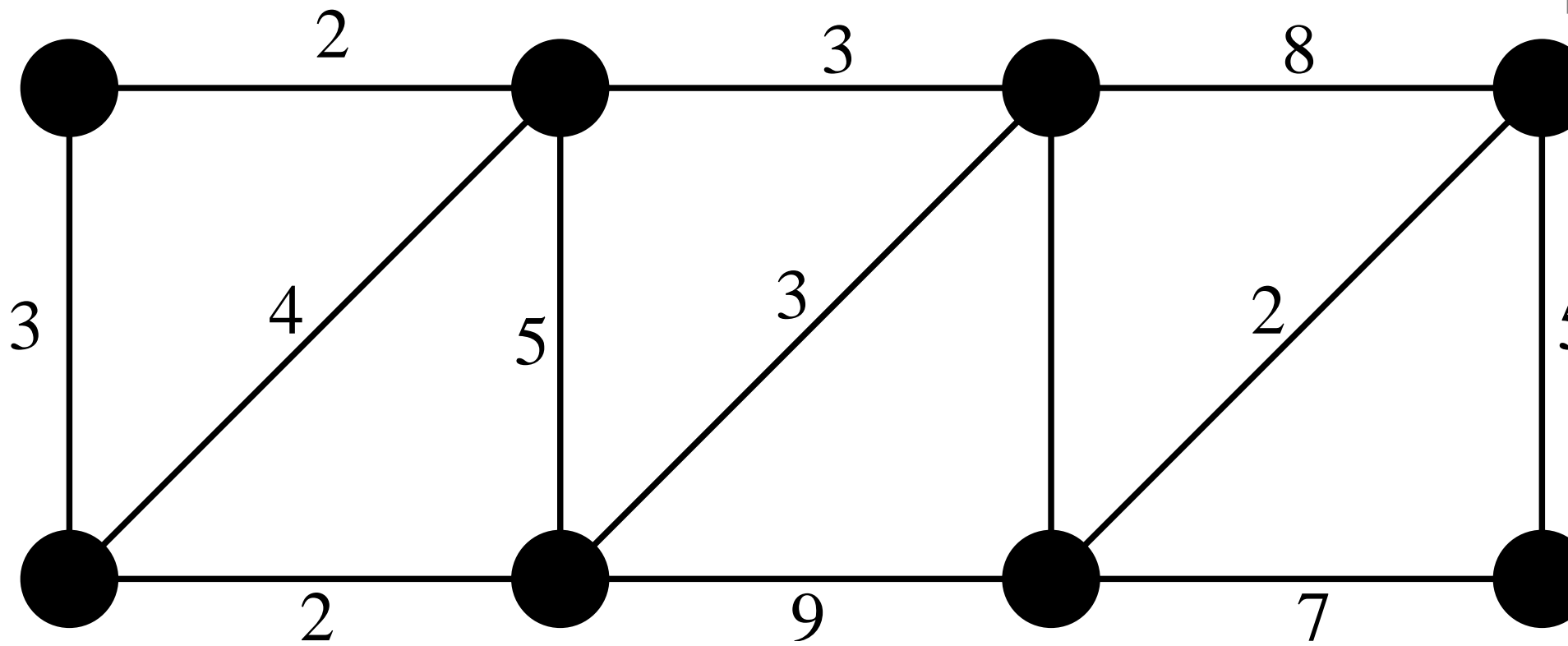
- W przypadku, gdy graf G ma **dokładnie dwa** wierzchołki u i v stopnia nieparzystego, to rozwiązanie sprowadza się do znalezienia (u, v) -ścieżki o minimalnej wadze i duplikacji jej krawędzi.

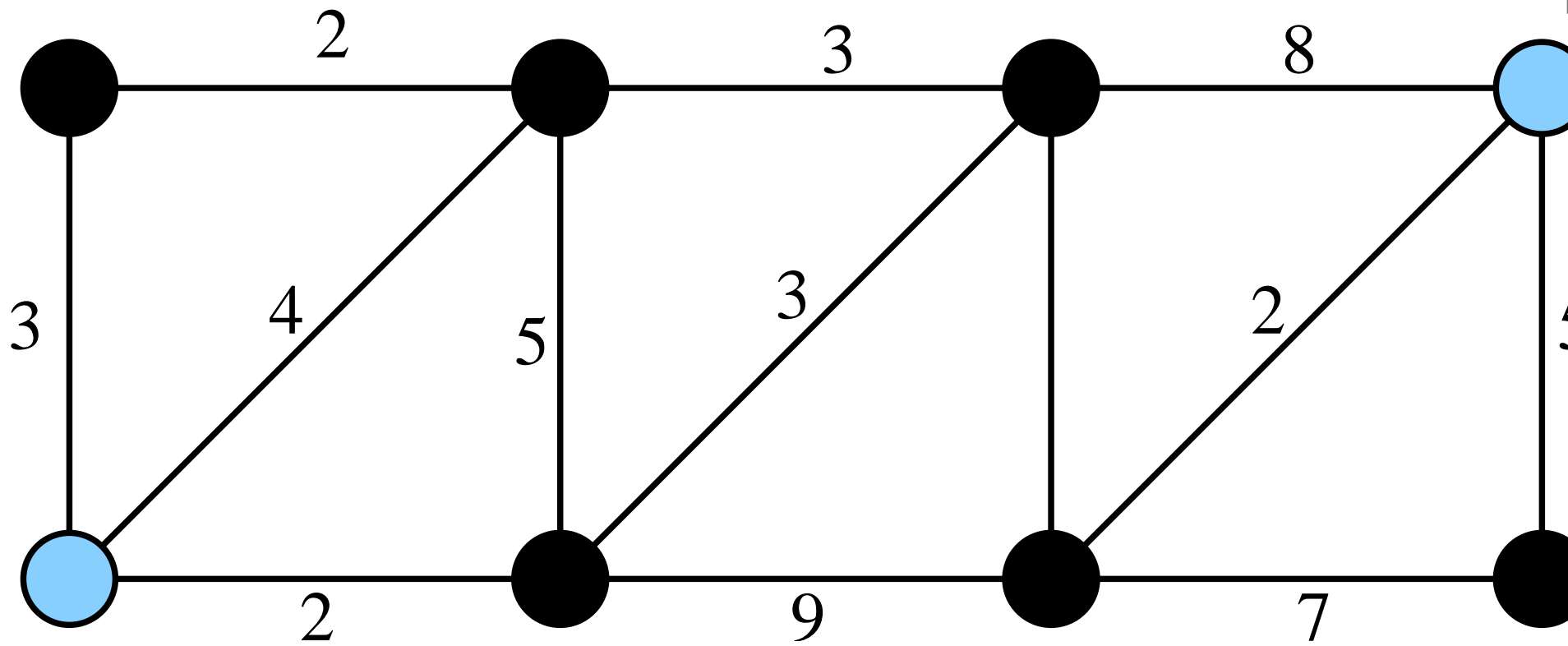
Algorytm Edmondsa-Johnsona

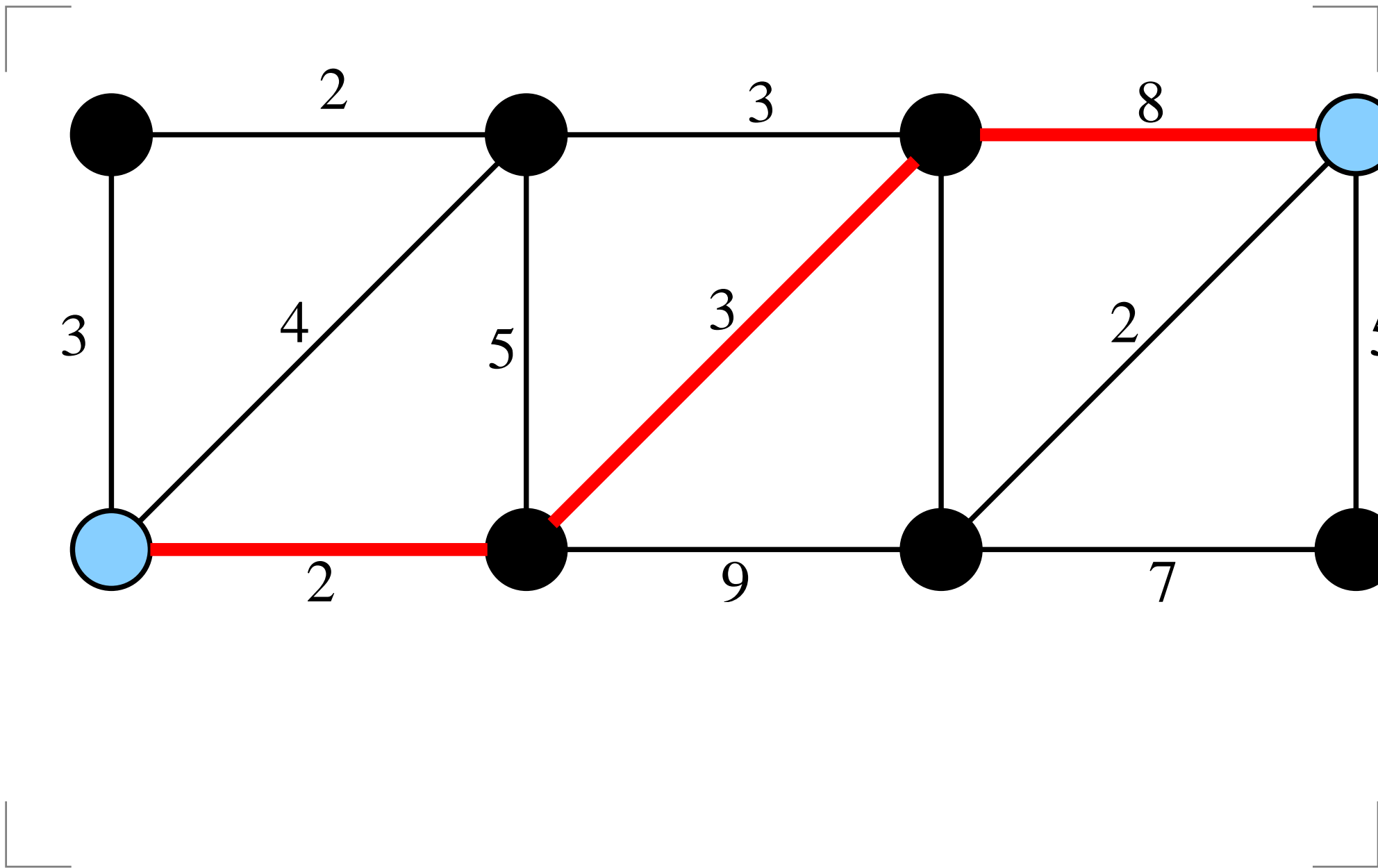
1. Korzystając z algorytmu znajdowania najkrótszych ścieżek wyznaczamy macierz $D(G) = (d_{ij})$ wymiaru $|V^-| \times |V^-|$, gdzie d_{ij} jest odległością (wagą najkrótszej ścieżki) wierzchołków v_i i v_j (dla $v_i, v_j \in V^-$).
2. Na podstawie macierzy D znajdujemy skojarzenie ścieżkowe \mathcal{M} – zbiór ścieżek kojarzących w pary wierzchołki z V^- , takich że suma wag krawędzi tych ścieżek jest minimalna (zauważmy, że w kroku tym szukamy optymalnego skojarzenia w grafie na zbiorze wierzchołków V^- o macierzy wag D);
3. Duplikujemy krawędzie z \mathcal{M} otrzymując graf $G^+(\mathcal{M})$.
4. Do grafu $G^+(\mathcal{M})$ stosujemy algorytm Fleury'ego (lub algorytm Hierholzera).

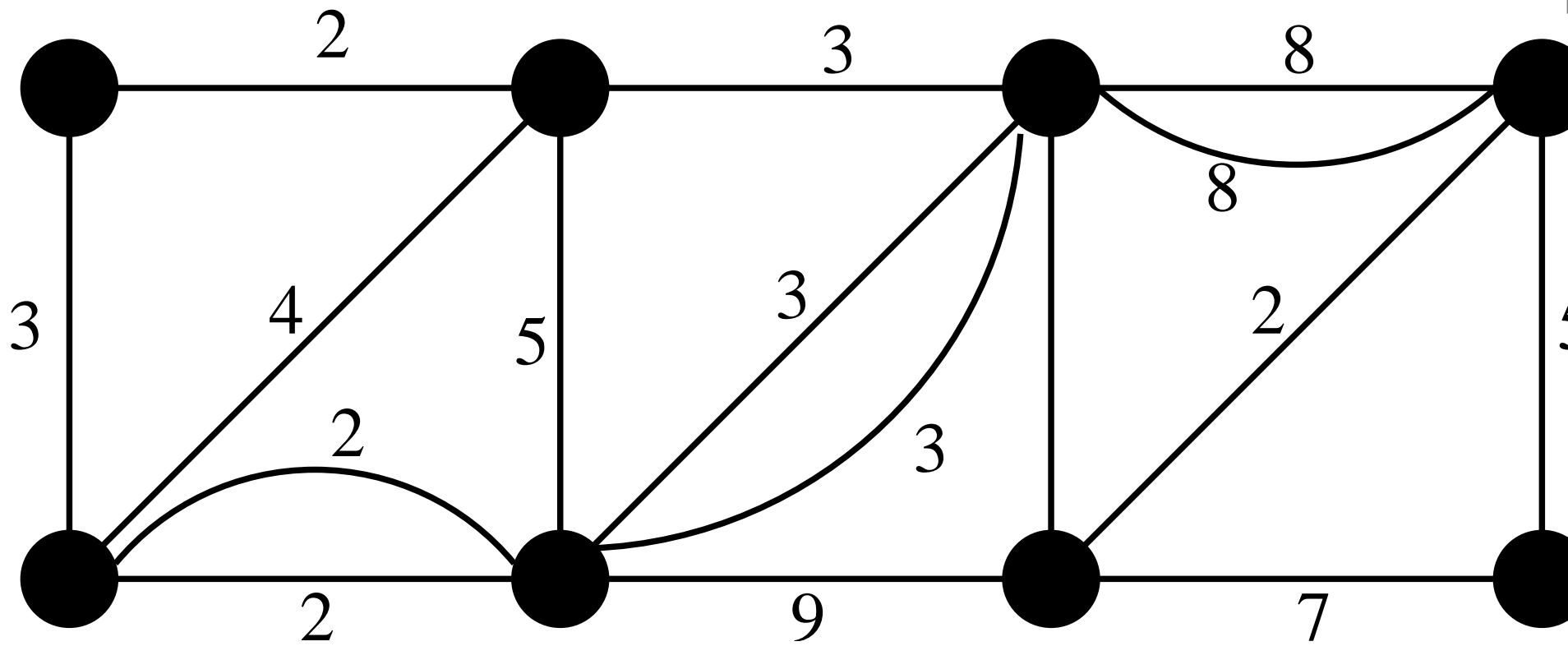
Przykład . Rozwiąż Problem Chińskiego Listonosza dla grafu z natępującą macierzą wag

$$W(G) = \begin{matrix} & \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{matrix} & \begin{pmatrix} \infty & 2 & \infty & \infty & \infty & \infty & \infty & 2 \\ 2 & \infty & 3 & \infty & \infty & \infty & 5 & 4 \\ \infty & 3 & \infty & 8 & \infty & 6 & 3 & \infty \\ \infty & \infty & 8 & \infty & 5 & 2 & \infty & \infty \\ \infty & \infty & \infty & 5 & \infty & 7 & \infty & \infty \\ \infty & \infty & 6 & 2 & 7 & \infty & 9 & \infty \\ \infty & 5 & 3 & \infty & \infty & 9 & \infty & 2 \\ 2 & 4 & \infty & \infty & \infty & \infty & 2 & \infty \end{pmatrix} \end{matrix} .$$









9. Cykle Hamiltona w grafie

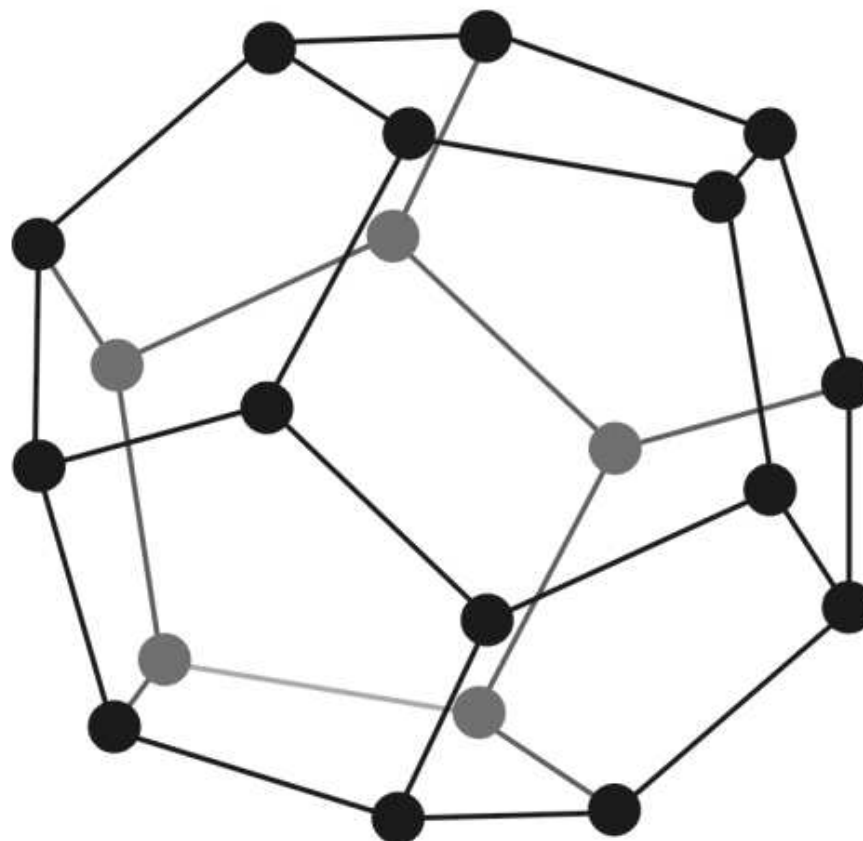
9.1 Definicje i twierdzenia

Definicja (Ścieżka i cykl Hamiltona). *Ścieżka zawierająca każdy wierzchołek $v \in V(G)$ jest nazywana ścieżką Hamiltona grafu G , podobnie, cykl Hamiltona jest to cykl, który zawiera wszystkie wierzchołki grafu G .*

Definicja (Graf Hamiltona). *Graf, który zawiera cykl Hamiltona nazywa się grafem Hamiltona (grafem hamiltonowskim).*

- *Sir William Rowan Hamilton (1805–1865) w 1856 roku opisał grę matematyczną na pewnym grafie na dwudziestu wierzchołkach – dwunastościanie foremny (patrz poniżej).*

- *W grze tej jedna osoba wpina pięć pinezek w pięciu kolejnych przyległych wierzchołkach dwunastościanu foremnego, a druga osoba musi uzupełnić tak utworzoną ścieżkę do rozpiętego cyklu.*



Twierdzenie . Jeżeli graf G jest hamiltonowski, to dla każdego niepustego podzbioru S zbioru wierzchołków $V(G)$ zachodzi

$$\omega(G - S) \leq |S|.$$

Dowód. Niech C będzie cyklem Hamiltona grafu G . Wtedy dla każdego niepustego podzbioru $S \subset V$

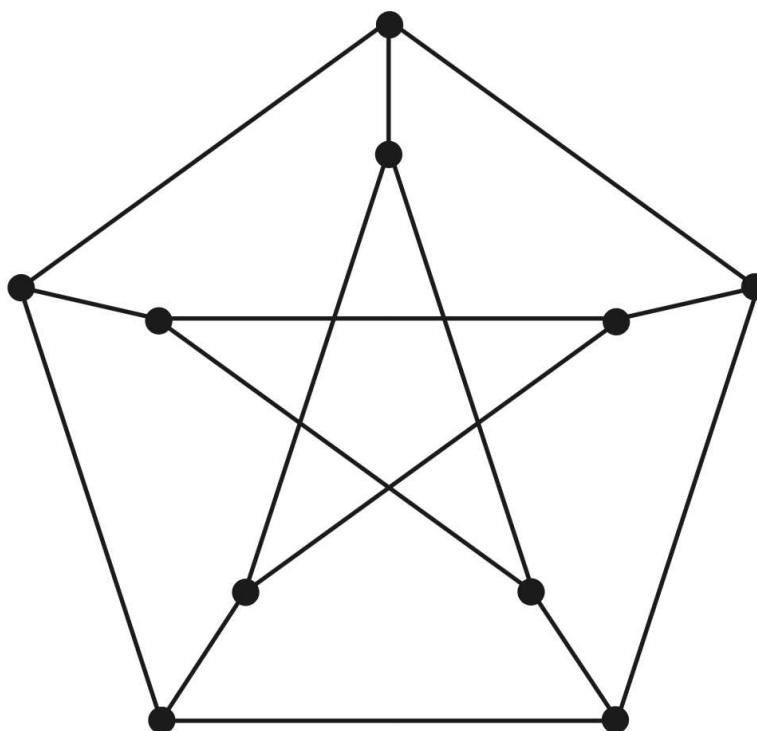
$$\omega(C - S) \leq |S|.$$

Ponieważ $C - S$ jest rozpiętym podgrafem grafu $G - S$, mamy również

$$\omega(G - S) \leq \omega(C - S) \leq |S|,$$

co kończy dowód. 

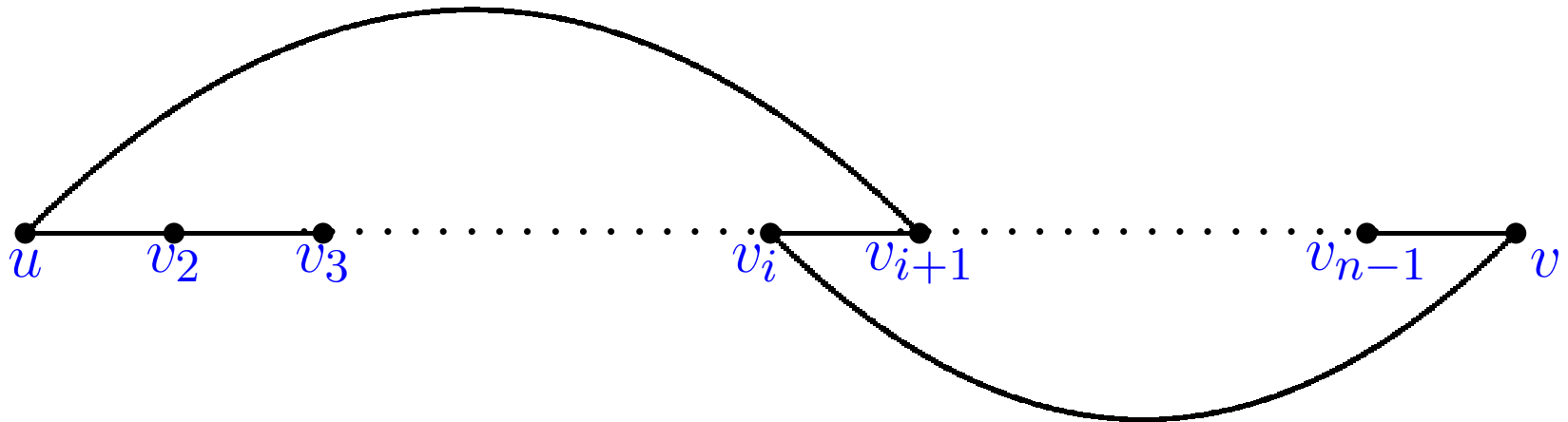
- *Powyższe twierdzenie może służyć nam do pokazania, że dany graf nie jest hamiltonowski. Jednakże ta metoda nie zawsze może być zastosowana (nie jest to bowiem warunek dostateczny). Nie pomoże nam ona, na przykład, w przypadku grafu Petersena który nie jest hamiltonowski ale spełnia warunki twierdzenia.*



Twierdzenie (Ore). Niech graf G będzie n wierzchołkowym grafem prostym oraz niech u i v będą nieprzyległymi wierzchołkami grafu G , dla których $d_G(u) + d_G(v) \geq n$. Wówczas G jest hamiltonowski wtedy i tylko wtedy, gdy $G + uv$ jest hamiltonowski.

Dowód. Zauważmy, że rozpięty nadgraf dowolnego grafu Hamiltona jest hamiltonowski.

Przypuśćmy, że $G + uv$ jest hamiltonowski a graf G takim grafem nie jest. Wówczas każdy cykl Hamiltona w $G + uv$ musi zawierać krawędź uv . Zatem istnieje ścieżka Hamiltona $u = v_1, v_2, \dots, v_n = v$ w G . Zauważmy, że nie może istnieć takie i , dla którego $uv_{i+1} \in E(G)$ oraz $v_i v \in E(G)$, oznaczałoby to bowiem istnienie cyklu Hamiltona w grafie G postaci $v_1 v_2, \dots, v_i, v_n, v_{n-1}, \dots, v_{i+1}, v_1$, co jest sprzeczne z naszym założeniem (patrz poniższy rysunek).



To oznacza, że v nie może być przyległy do $d_G(u) - 1$ wierzchołków występujących bezpośrednio przed sąsiadami wierzchołka u (nie licząc v_2), czyli

$$d_G(v) \leq n - 2 - (d_G(u) - 1),$$

co oczywiście przeczy założeniu, że $d_G(u) + d_G(v) \geq n$. ■

Twierdzenie (Dirac). Jeżeli graf G jest grafem prostym takim, że

$|V(G)| \geq 3$ oraz $\delta(G) \geq \frac{|V(G)|}{2}$, to G jest hamiltonowski.

9.2 Algorytmy poszukiwania cykli Hamiltona w grafie

Algorytm Robertsa-Floresa

Dane: graf skierowany $G = (V, E)$

Poszukiwane: wszystkie cykle Hamiltona w G

1. *Budujemy macierz następników: kolumny macierzy odpowiadają wierzchołkom i zawierają ich następniki w pewnej na początku ustalonej kolejności.*
2. *Rozpoczynamy z dowolnego wierzchołka v . Bierzemy pierwszy „dostępny” (jeszcze nie włączony do zbioru S wierzchołków budowanego cyklu) następnik z kolumny odpowiadającej v . Załóżmy, że jest to wierzchołek u . $S \leftarrow S \cup \{u\}$. Następnie bierzemy pierwszy dostępny następnik wierzchołka u z kolumny odpowiadającej u , itd.*

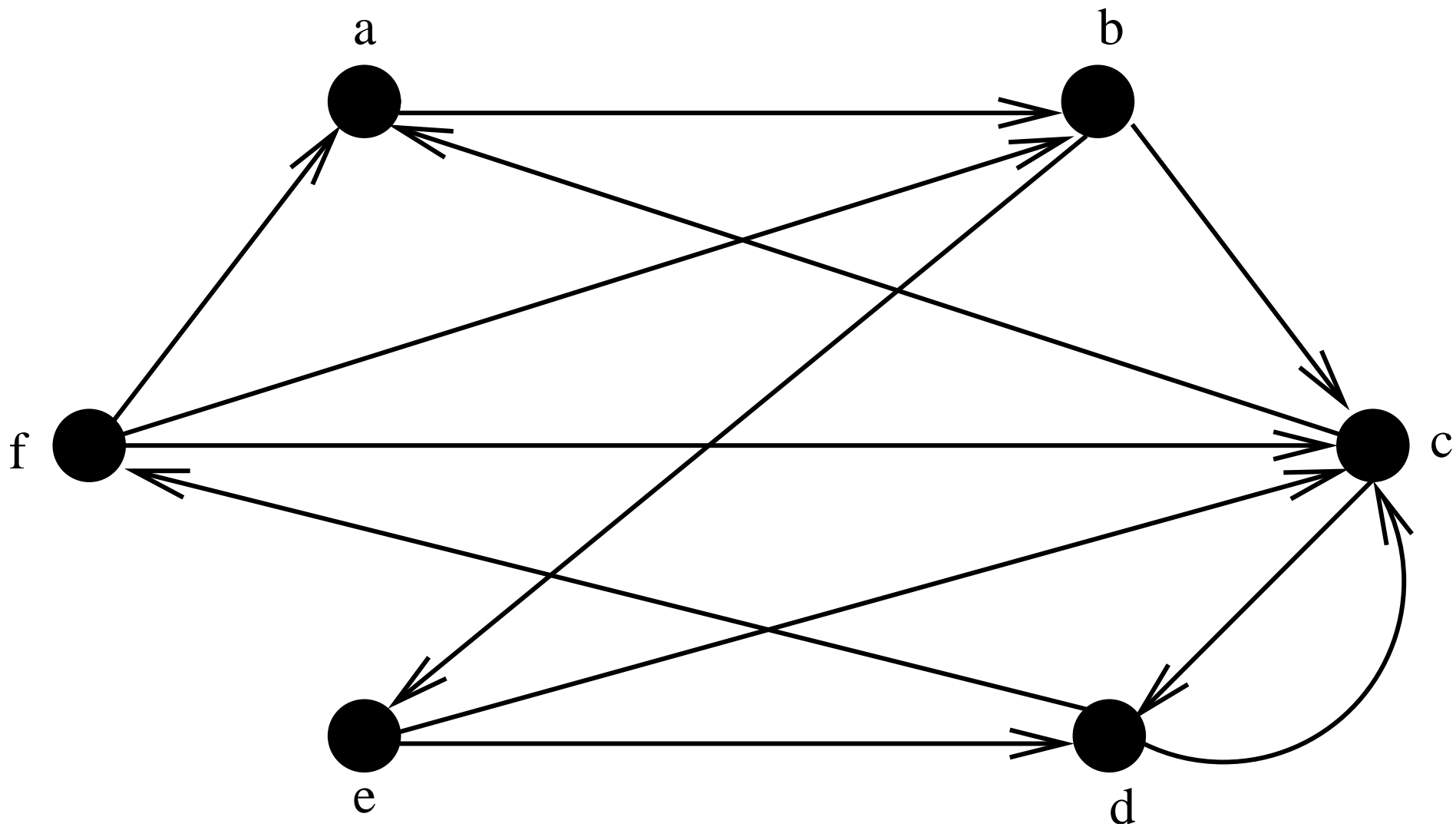
3. Mamy następujące możliwości:

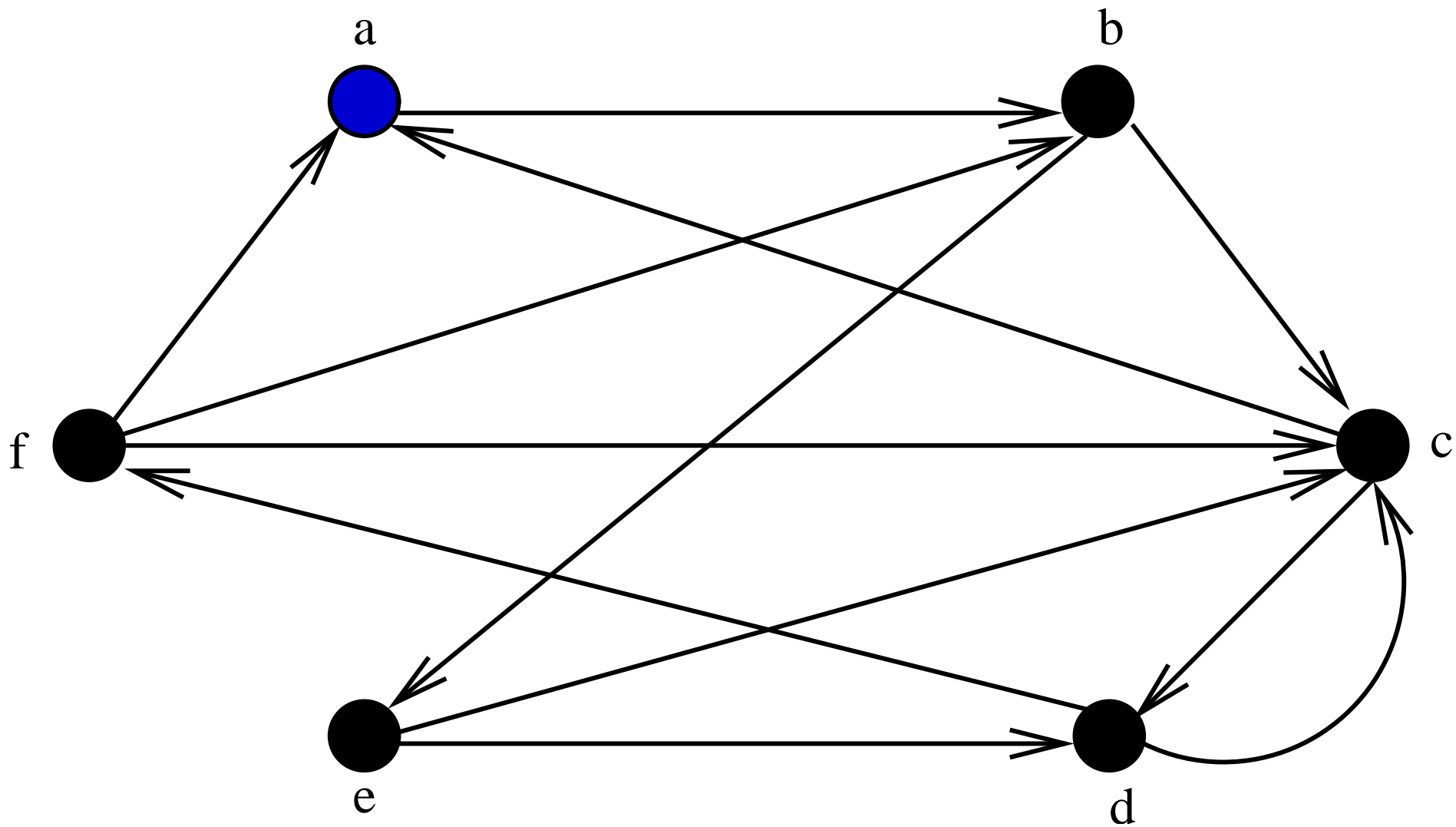
- (i) Nie ma dostępnego następnika - następuje krok powrotu - wyrzucamy z S ostatnio dodany wierzchołek, wracamy do kolumny, z której został on wybrany i bierzemy kolejny dostępny następnik; następnie bierzemy pierwszy dostępny jego następnik, itd. (oczywiście za każdym razem, gdy nie ma dostępnego następnika następuje krok powrotu);
- (ii) Zbiór S ma już moc n (czyli znaleźliśmy już ścieżkę Hamiltona H z v do w , gdzie w jest ostatnio dodanym do S wierzchołkiem). Sprawdzamy, czy istnieje łuk z w do v : jeśli TAK, to zapisujemy cykl Hamiltona $H + vw$ i krok powrotu (lub STOP jeśli chcemy znaleźć tylko jeden cykl Hamiltona); jeśli NIE, to krok powrotu.

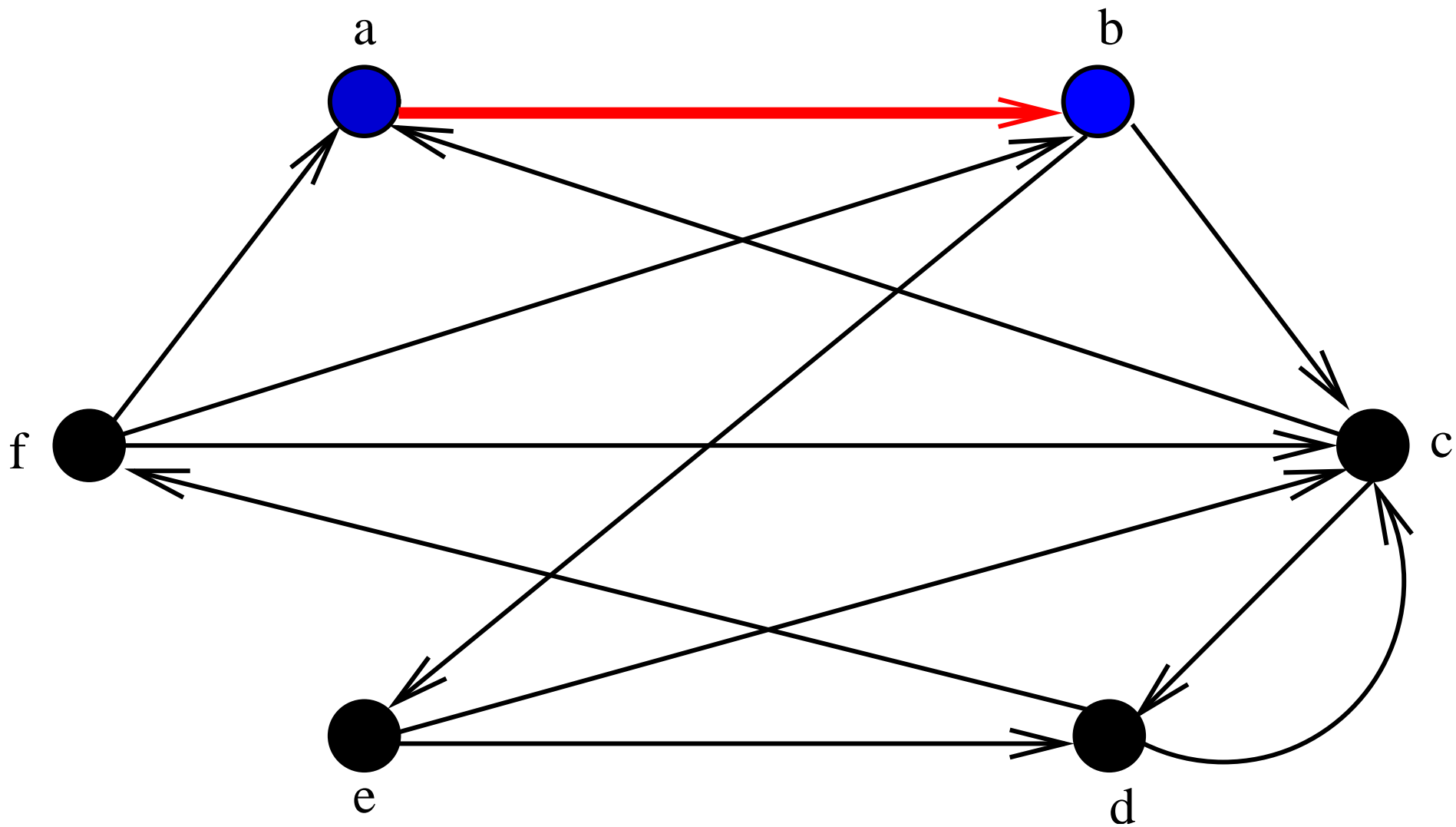
4. Algorytm kończy pracę, gdy powrócimy do wierzchołka v i nie ma już jego “dostępnych” następników.

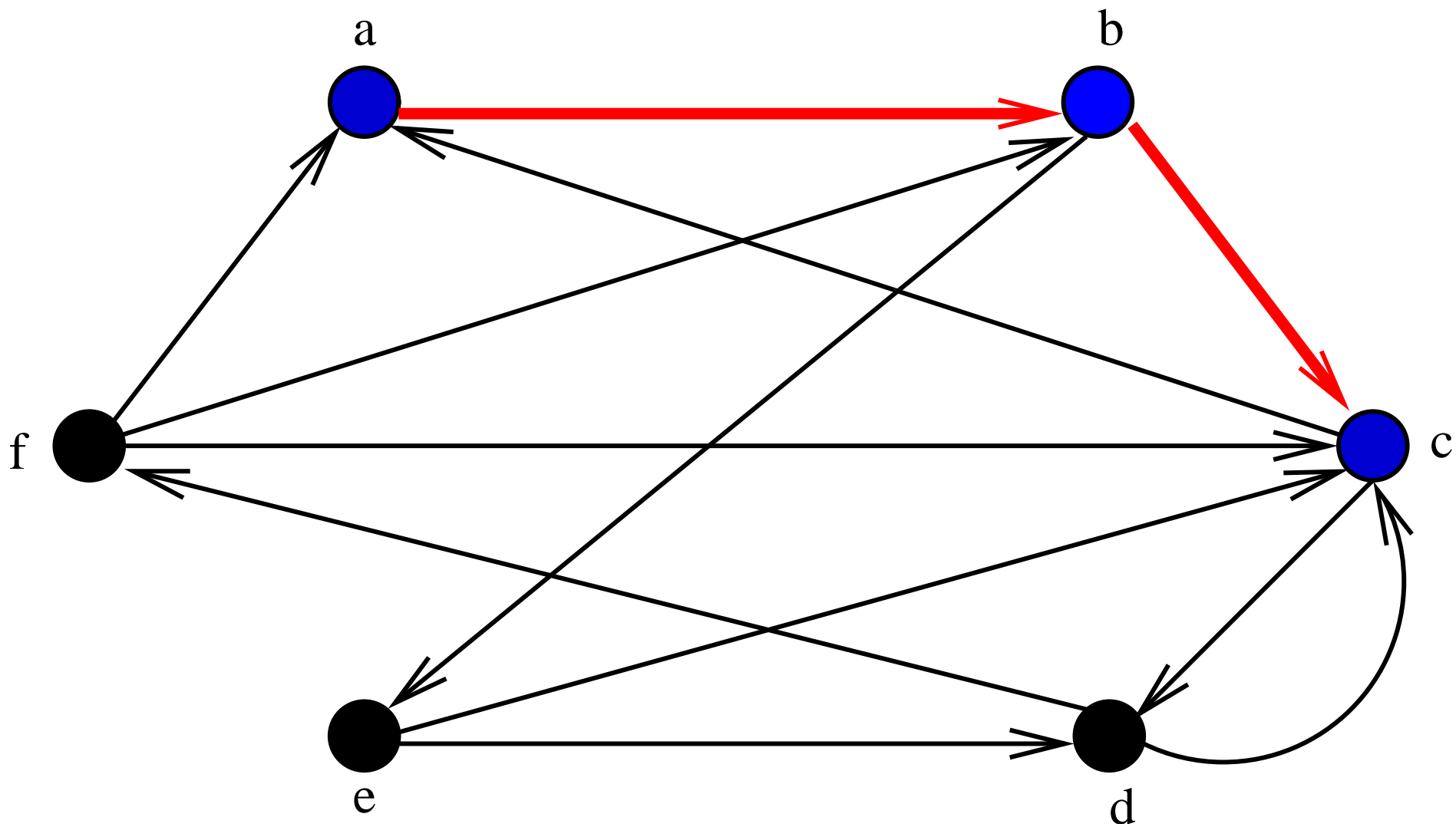
Przykład . Wyznaczyć za pomocą algorytmu Roberta-Floresa wszystkie cykle Hamiltona w grafie G o macierzy przejść:

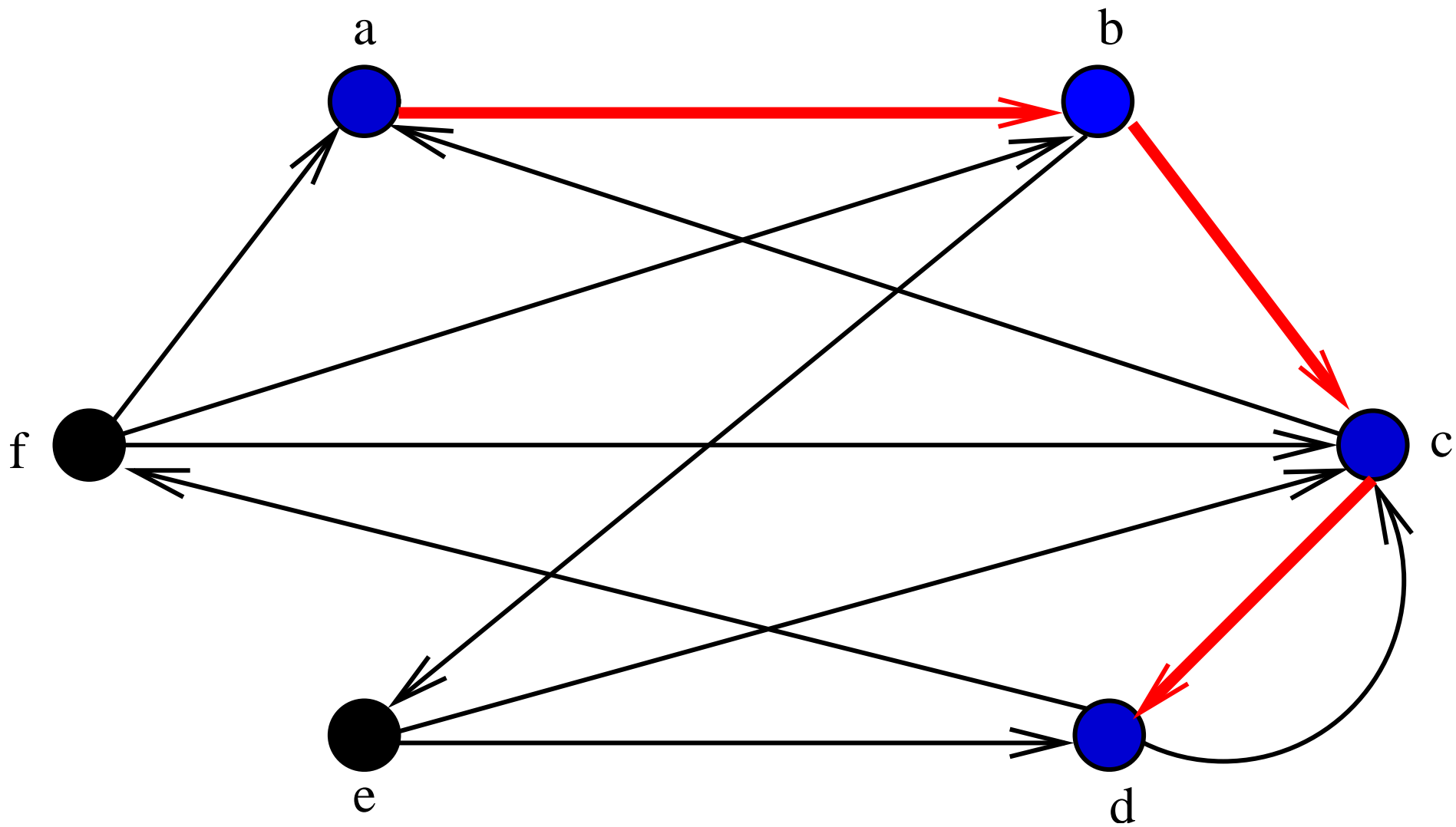
$$P(G) = \begin{pmatrix} \infty & 1 & \infty & \infty & \infty & \infty \\ \infty & \infty & 1 & \infty & 1 & \infty \\ 1 & \infty & \infty & 1 & \infty & \infty \\ \infty & \infty & 1 & \infty & \infty & 1 \\ \infty & \infty & 1 & 1 & \infty & \infty \\ 1 & 1 & 1 & \infty & \infty & \infty \end{pmatrix}.$$

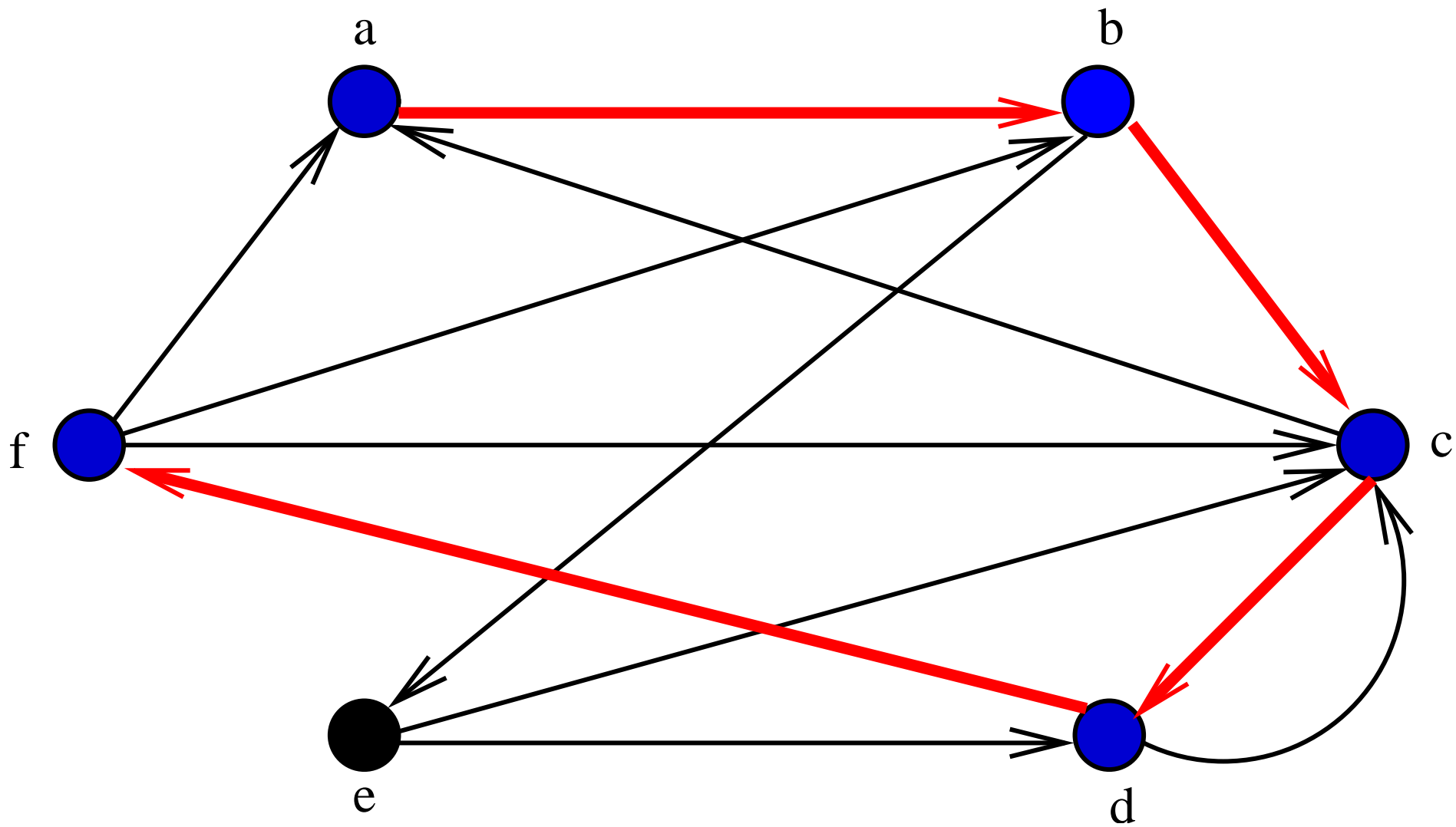


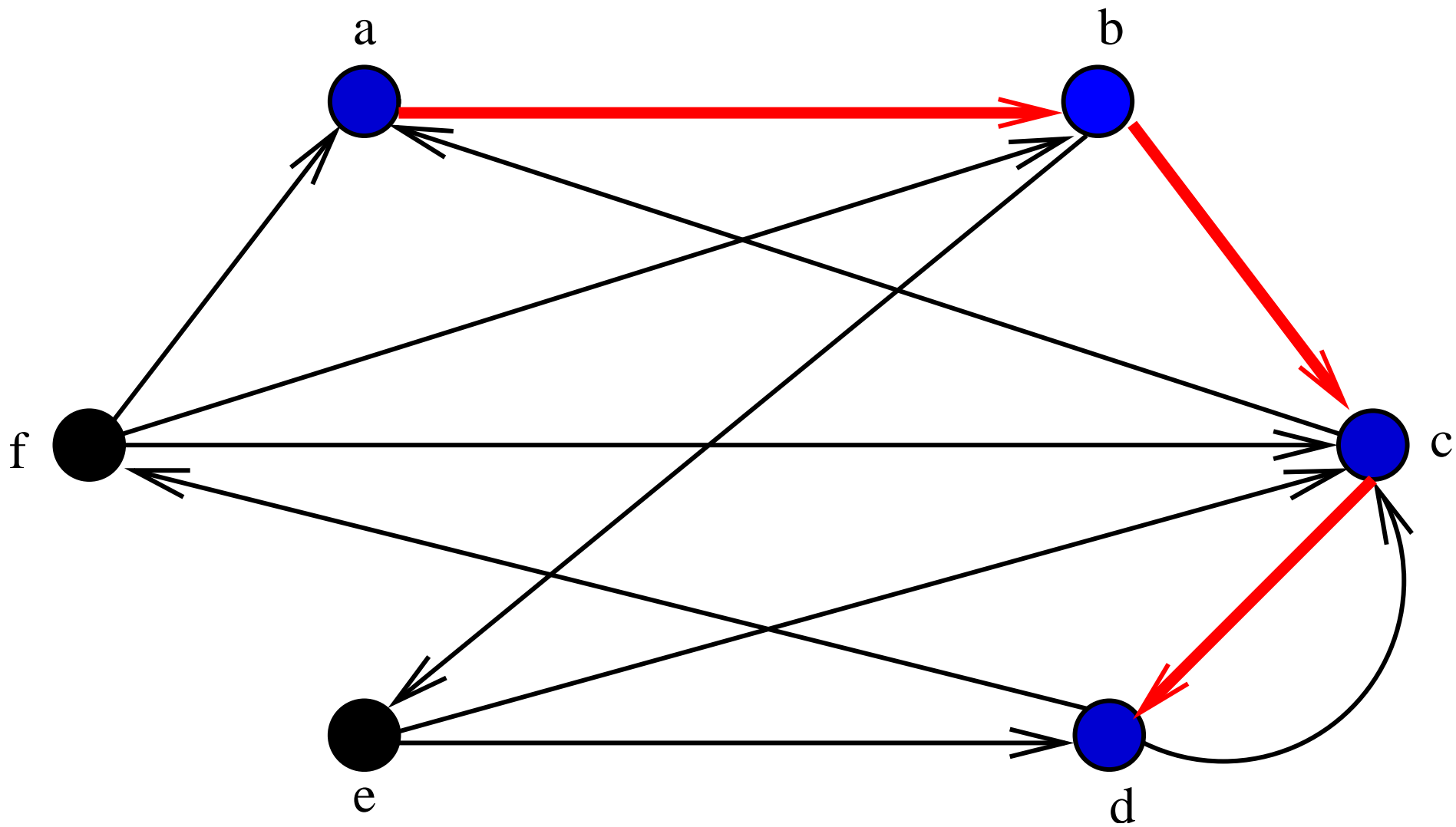


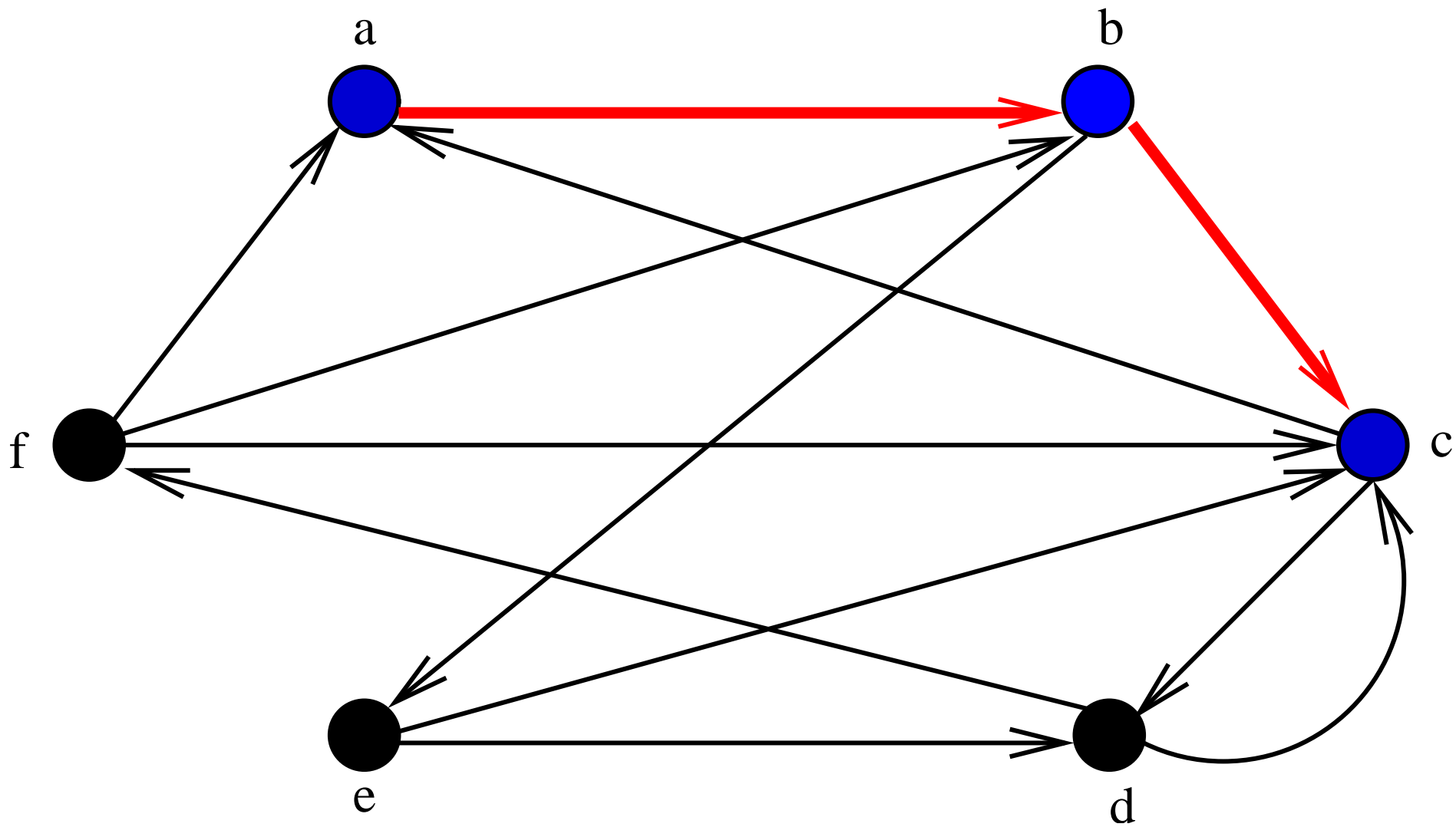


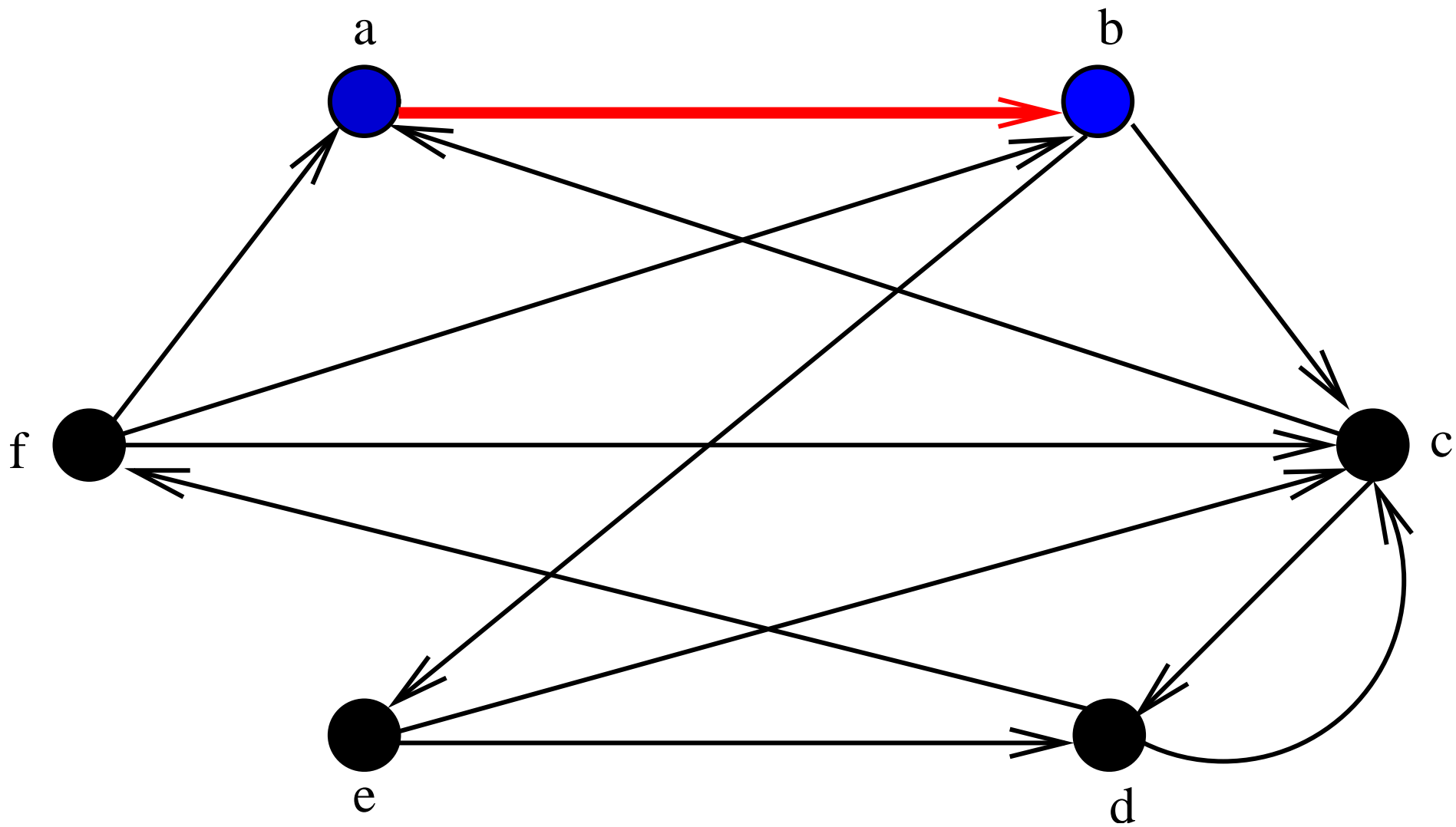


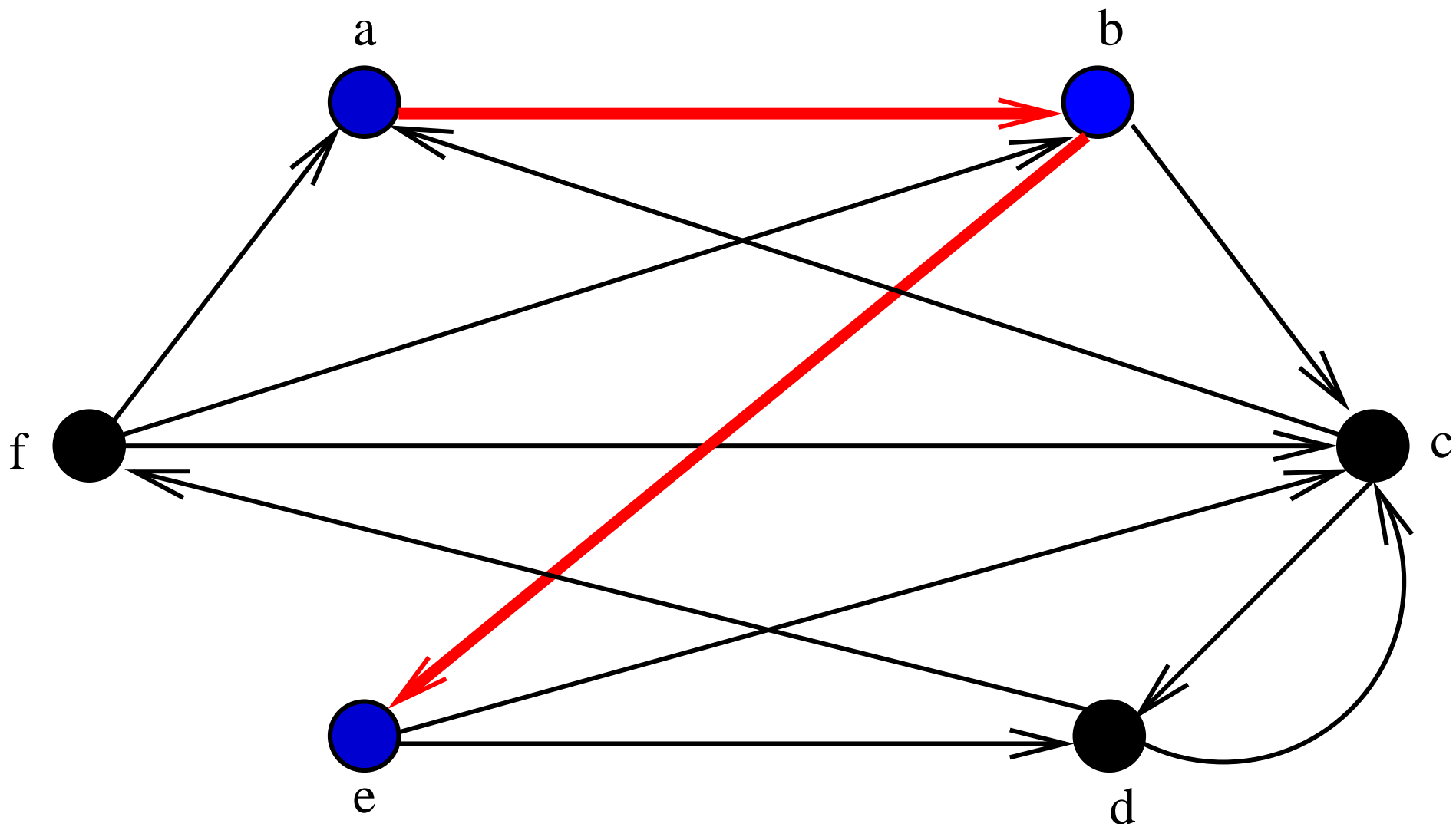


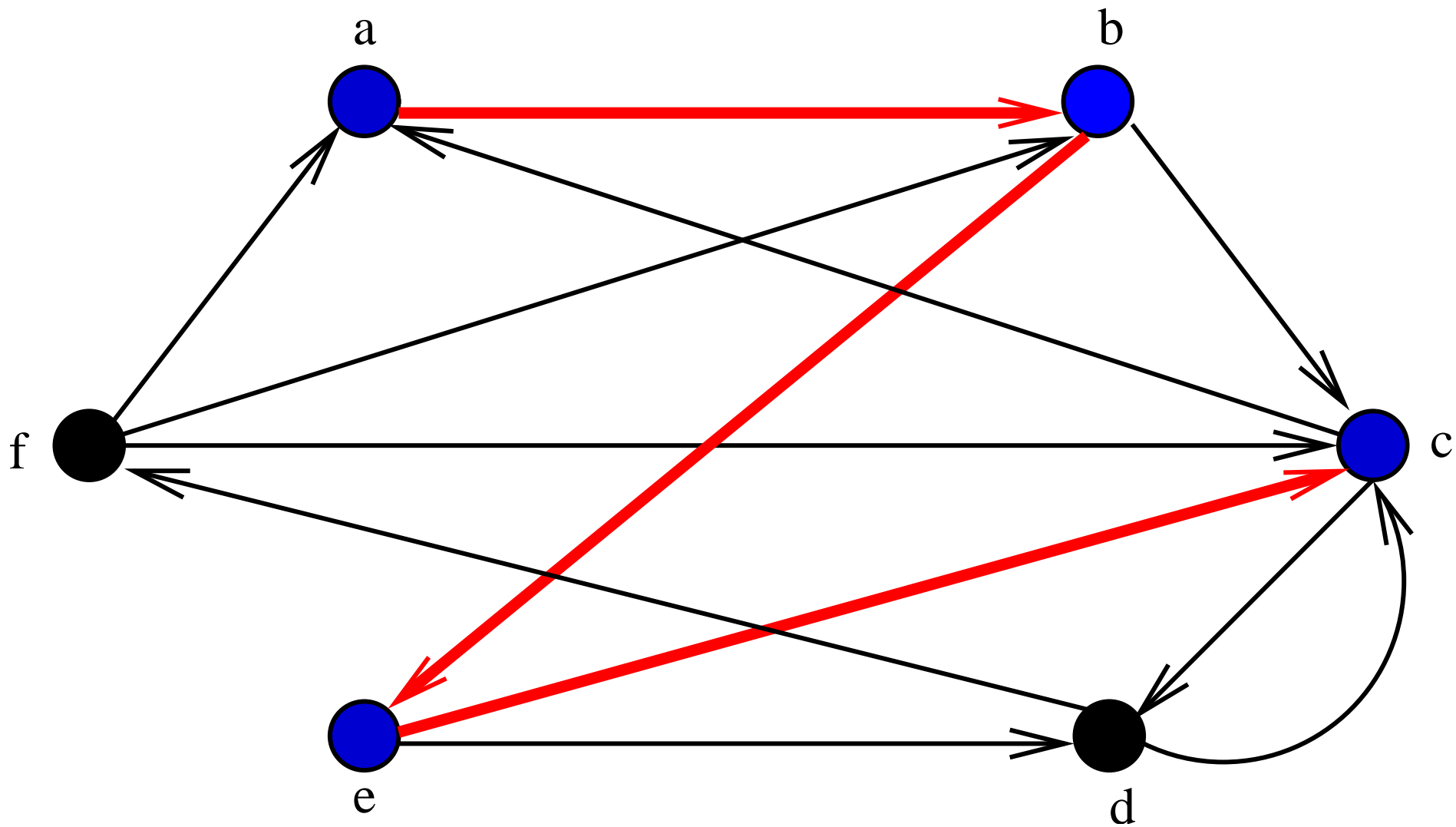


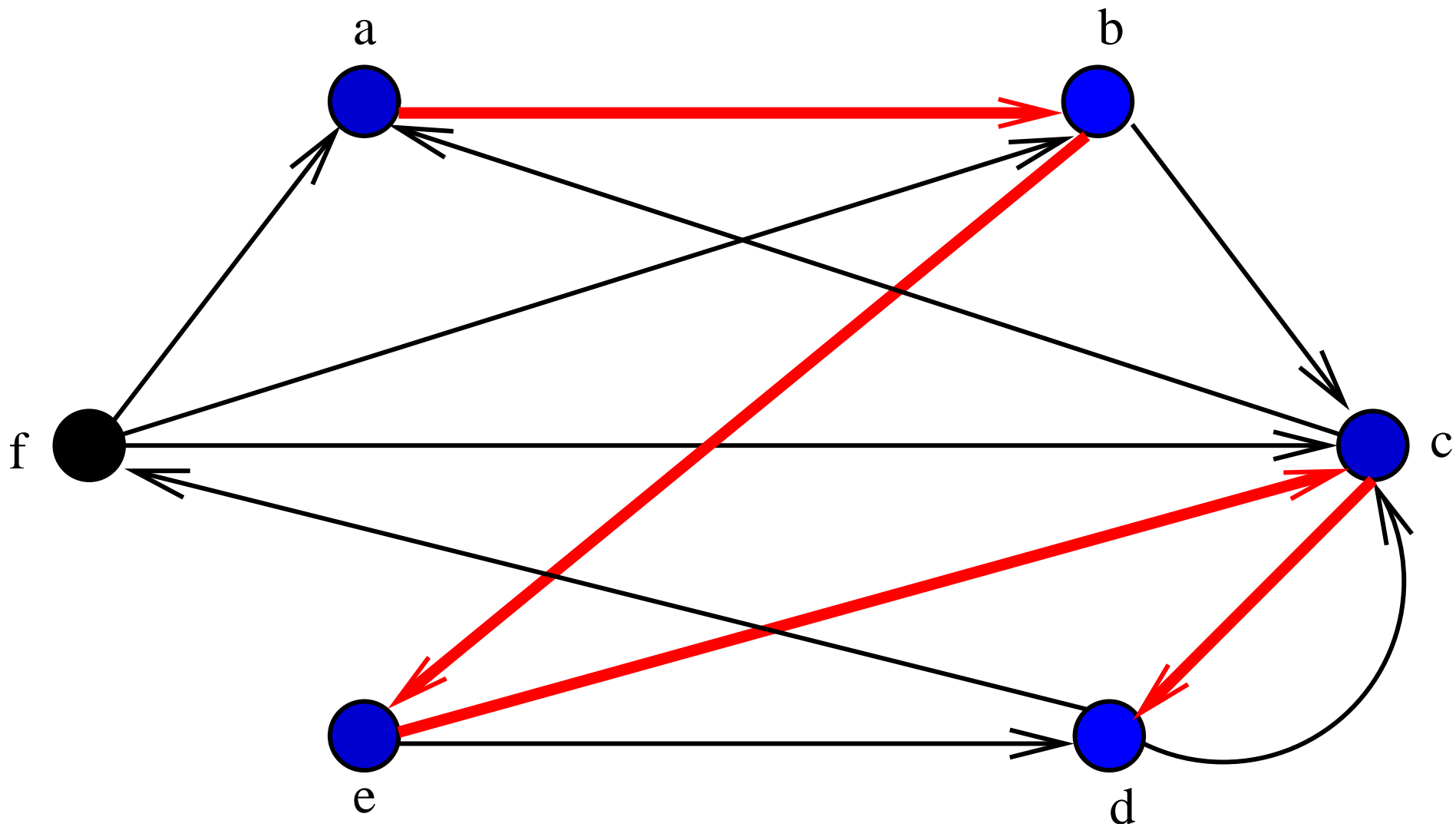


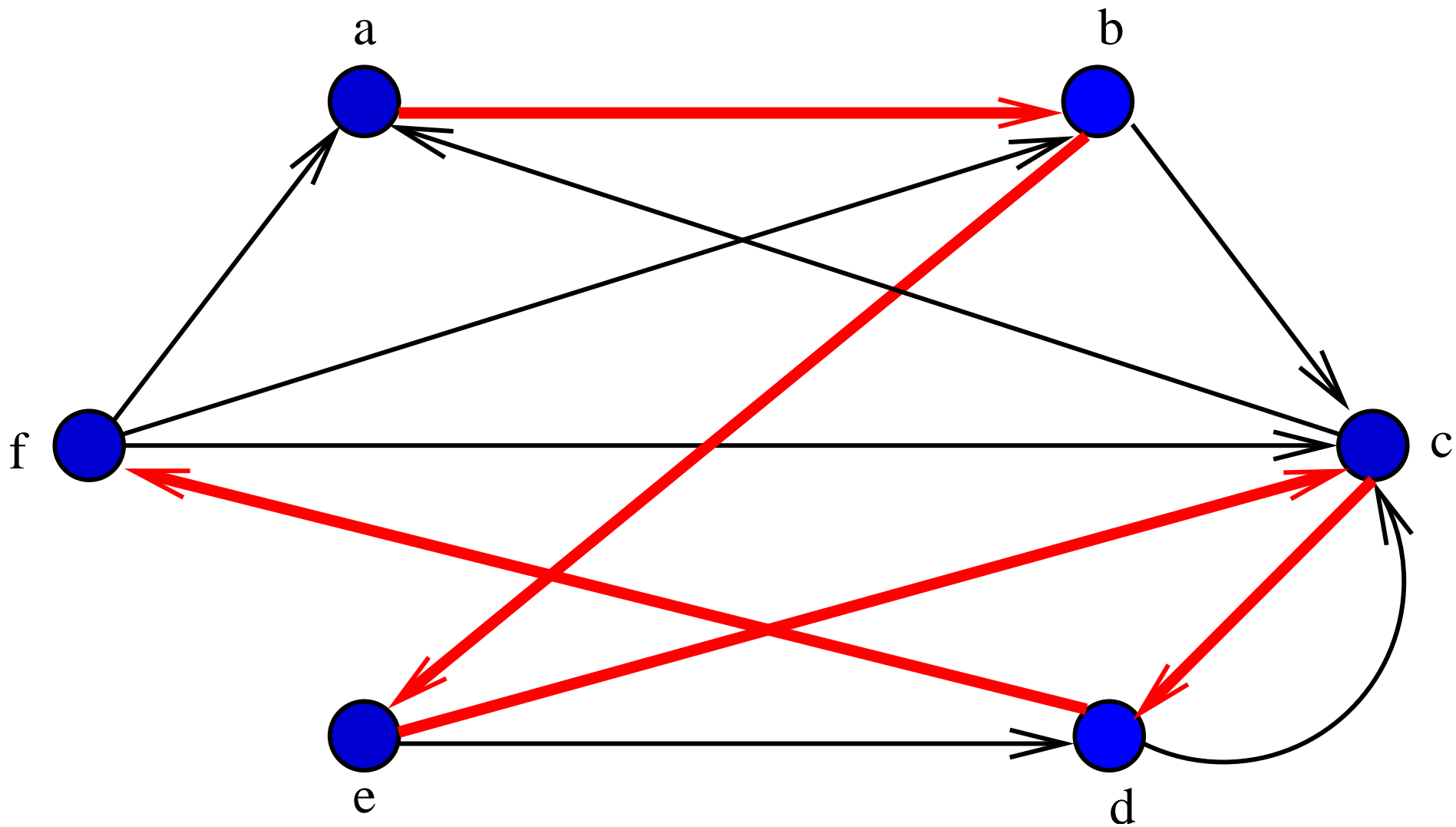


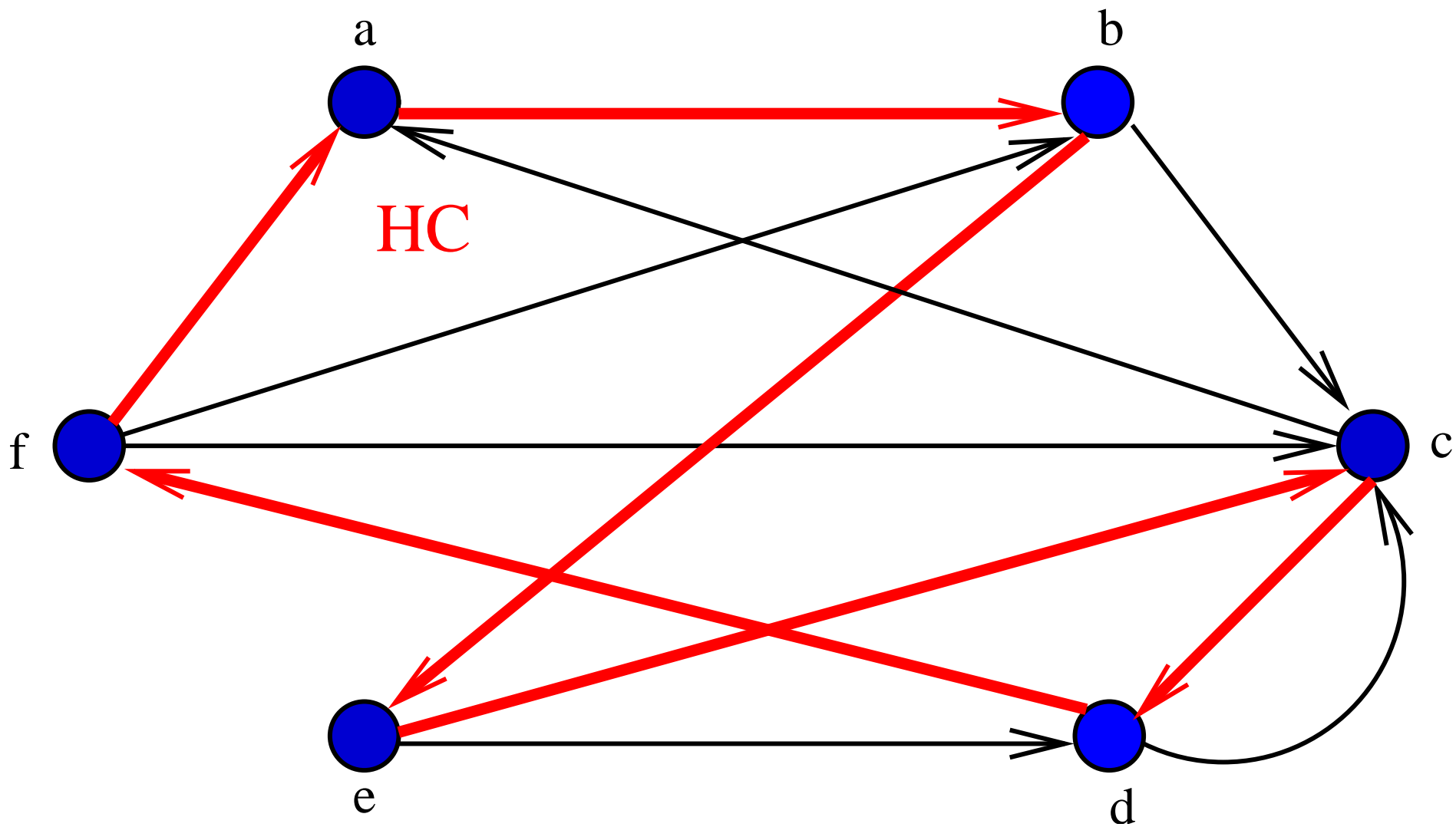


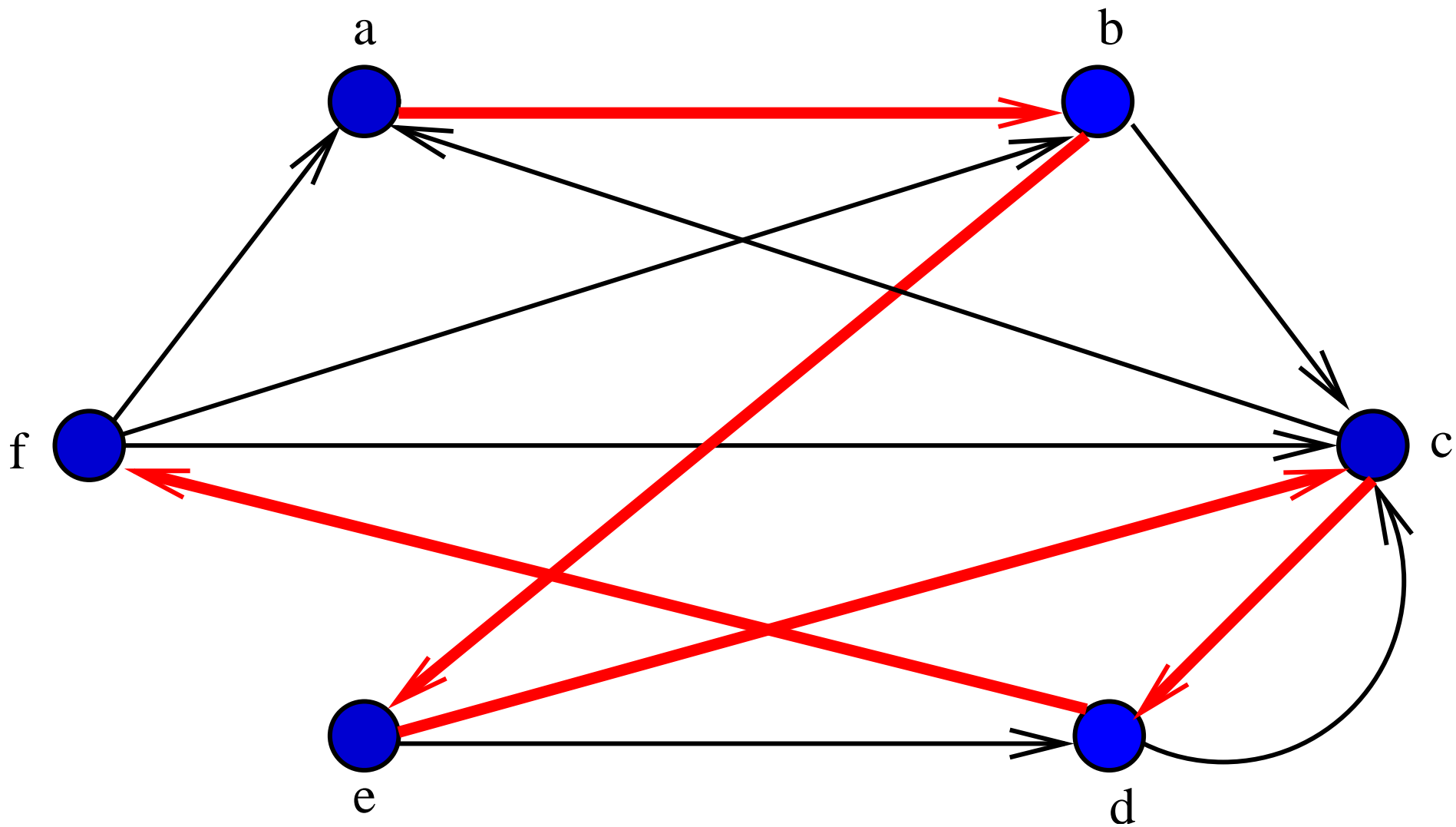


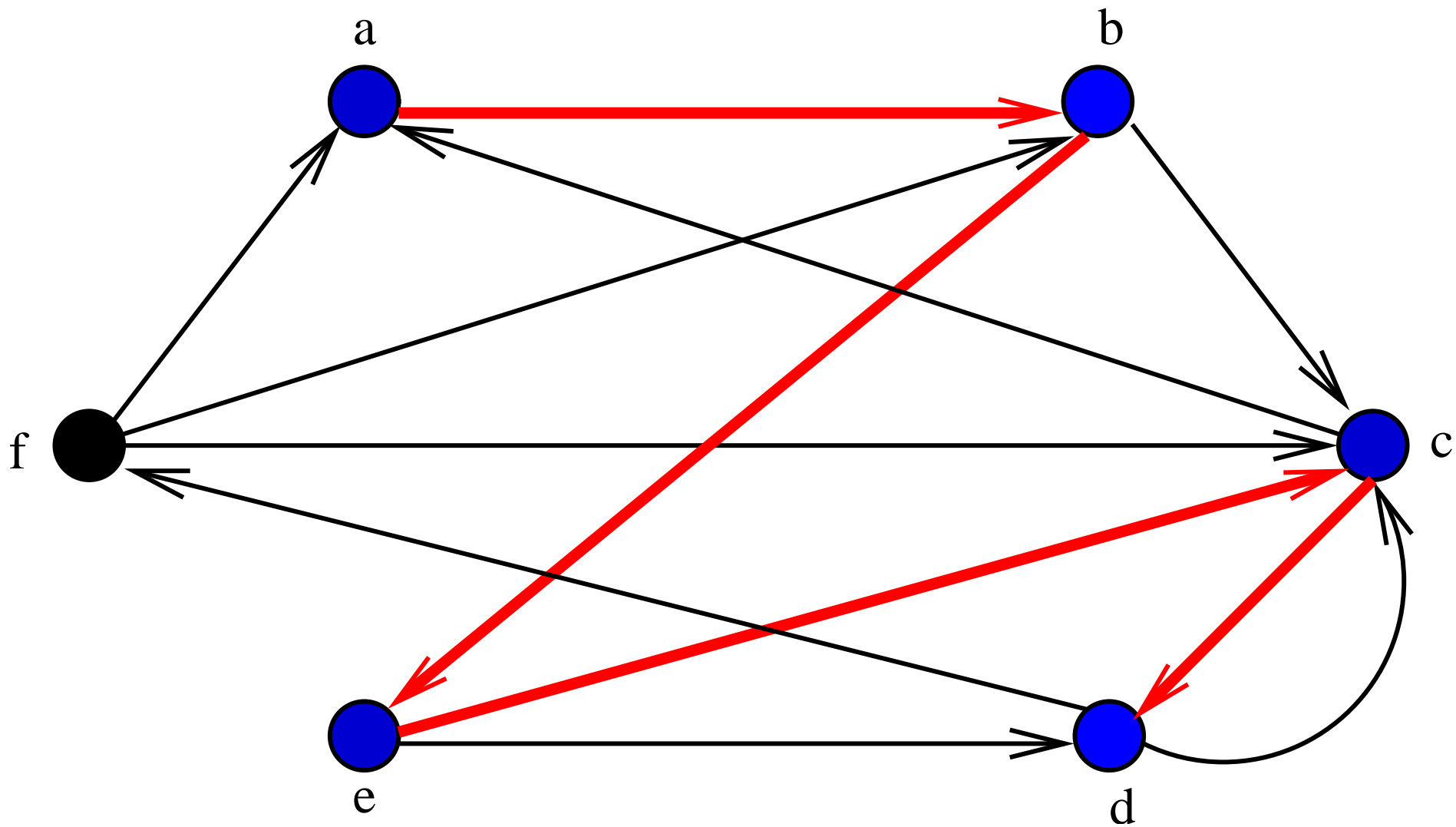


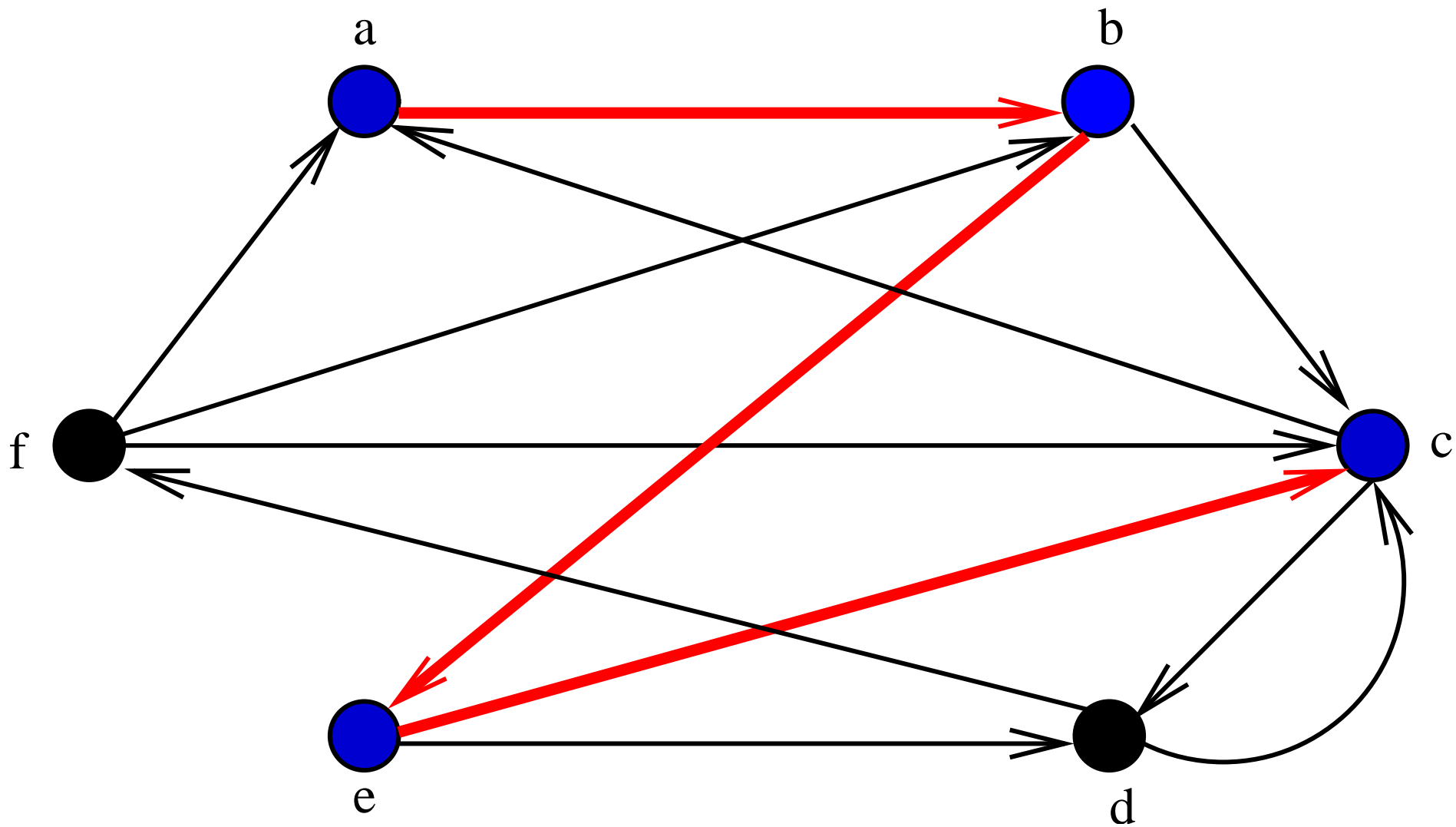


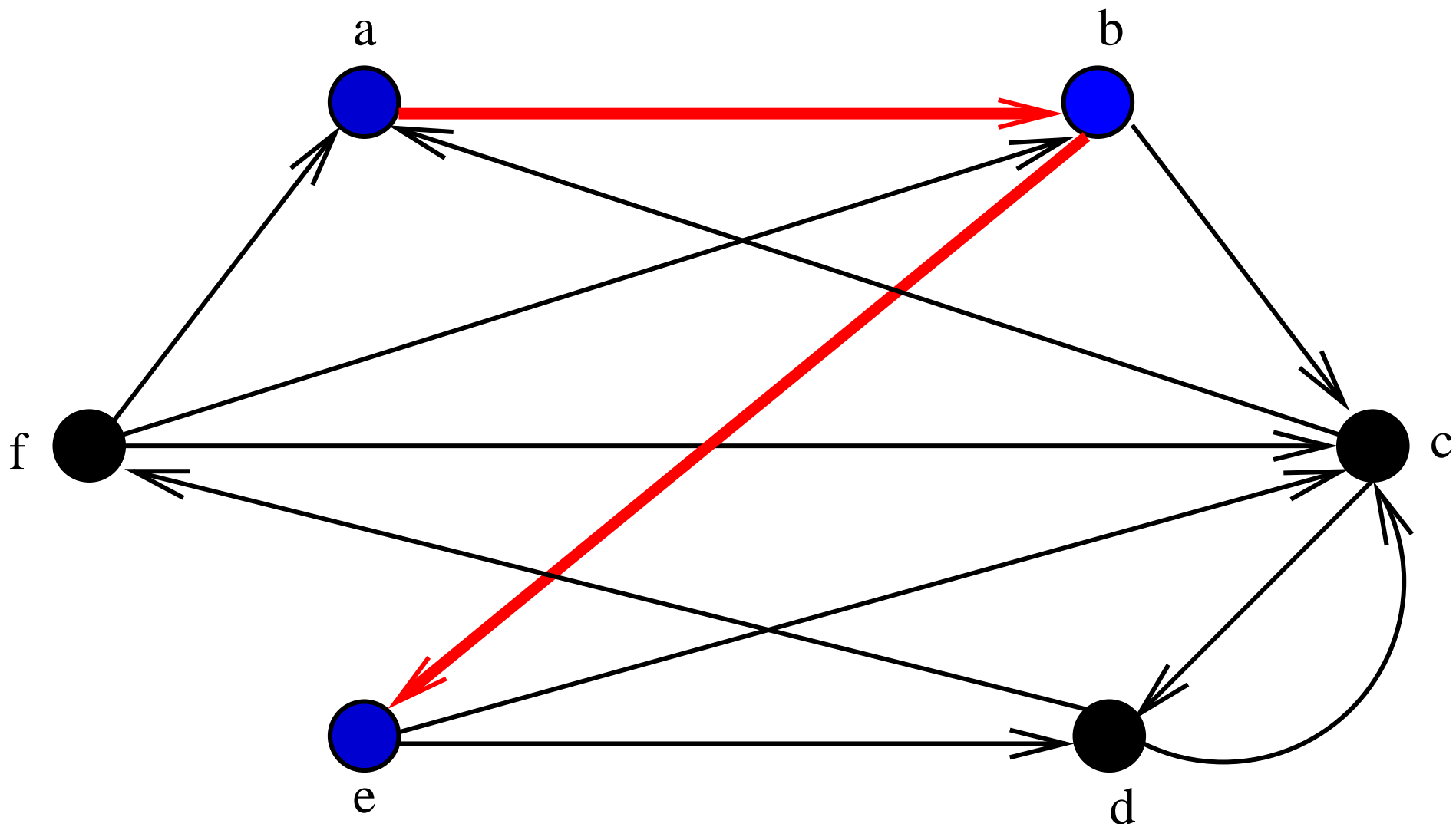


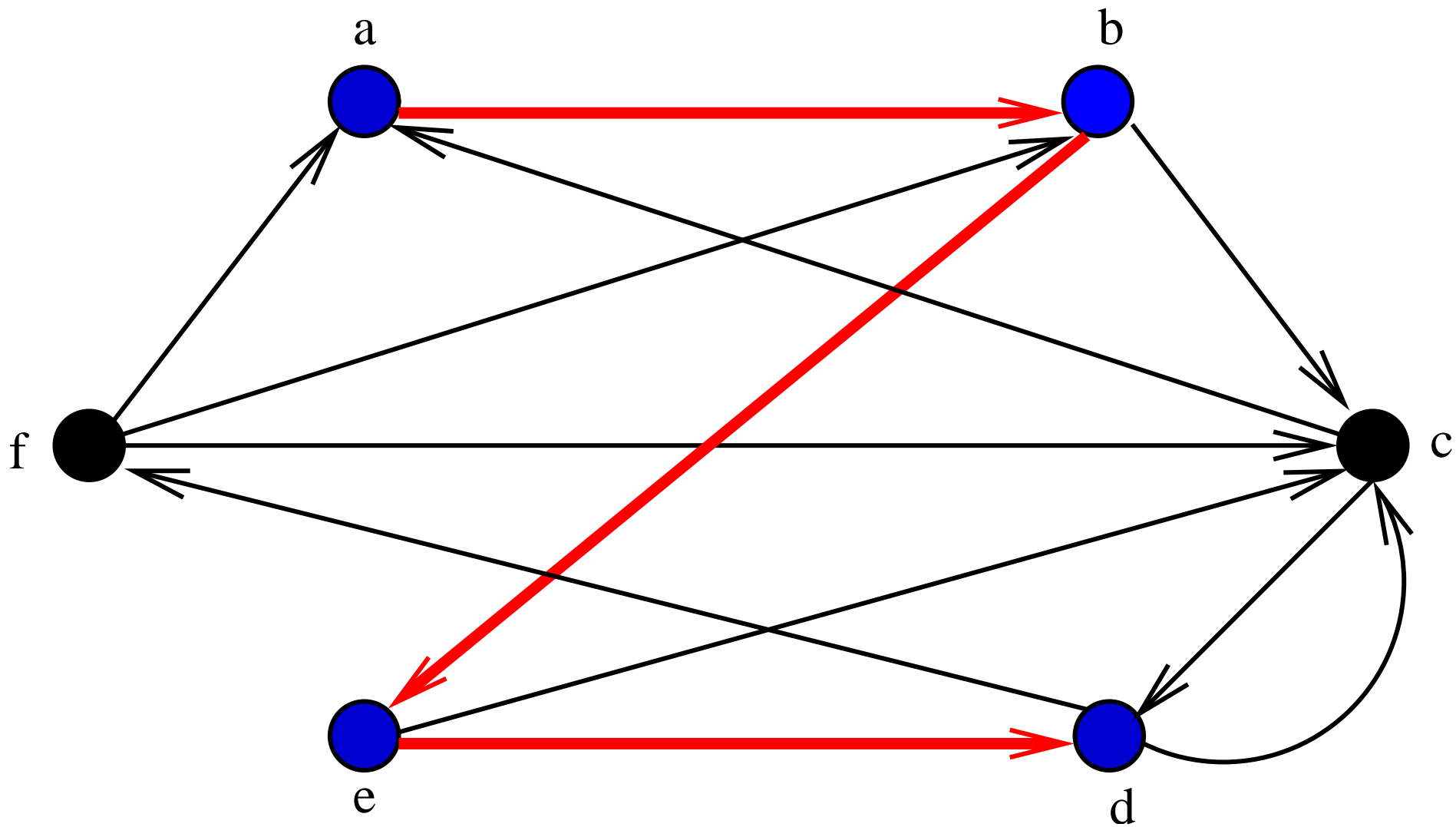


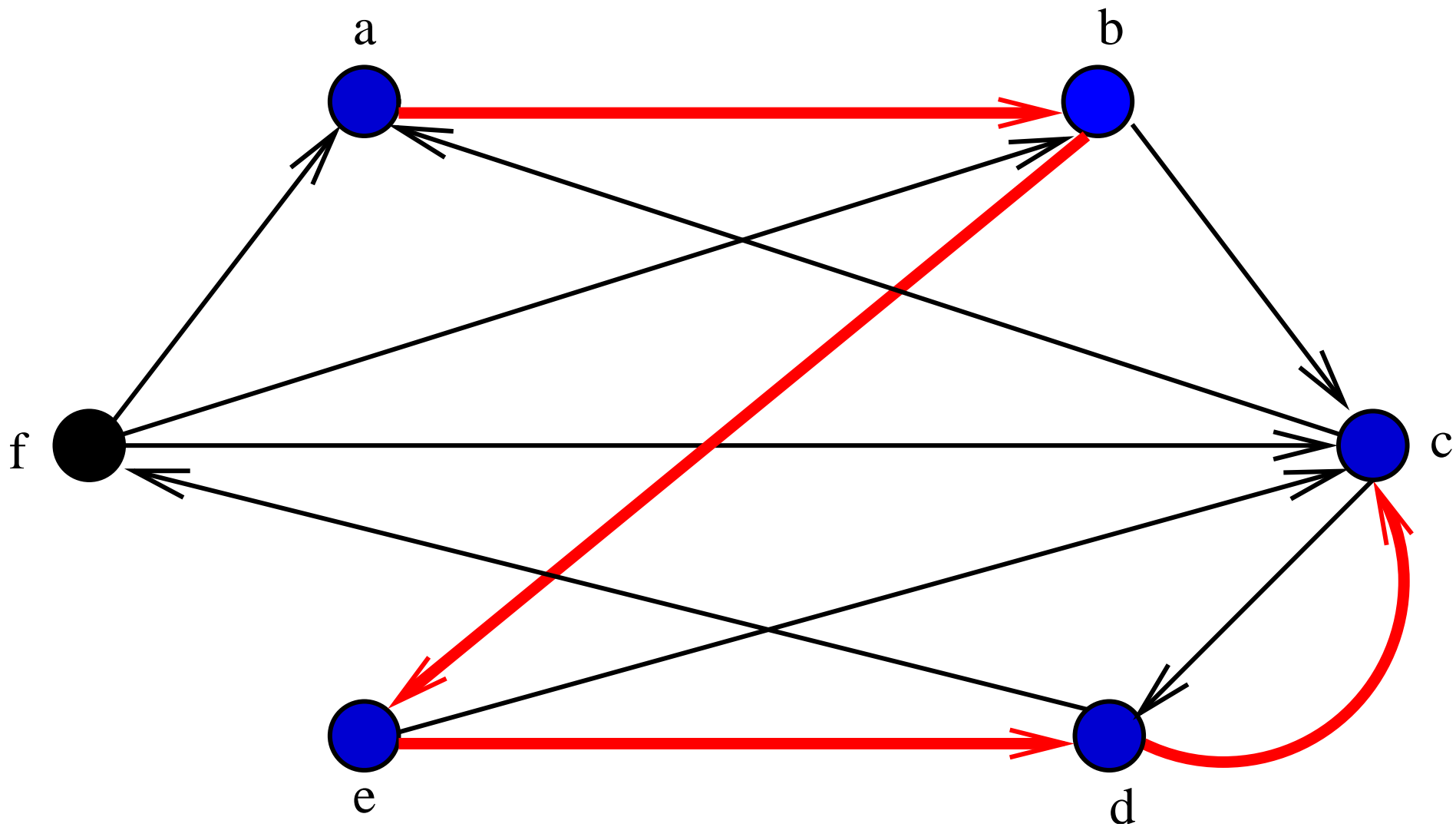


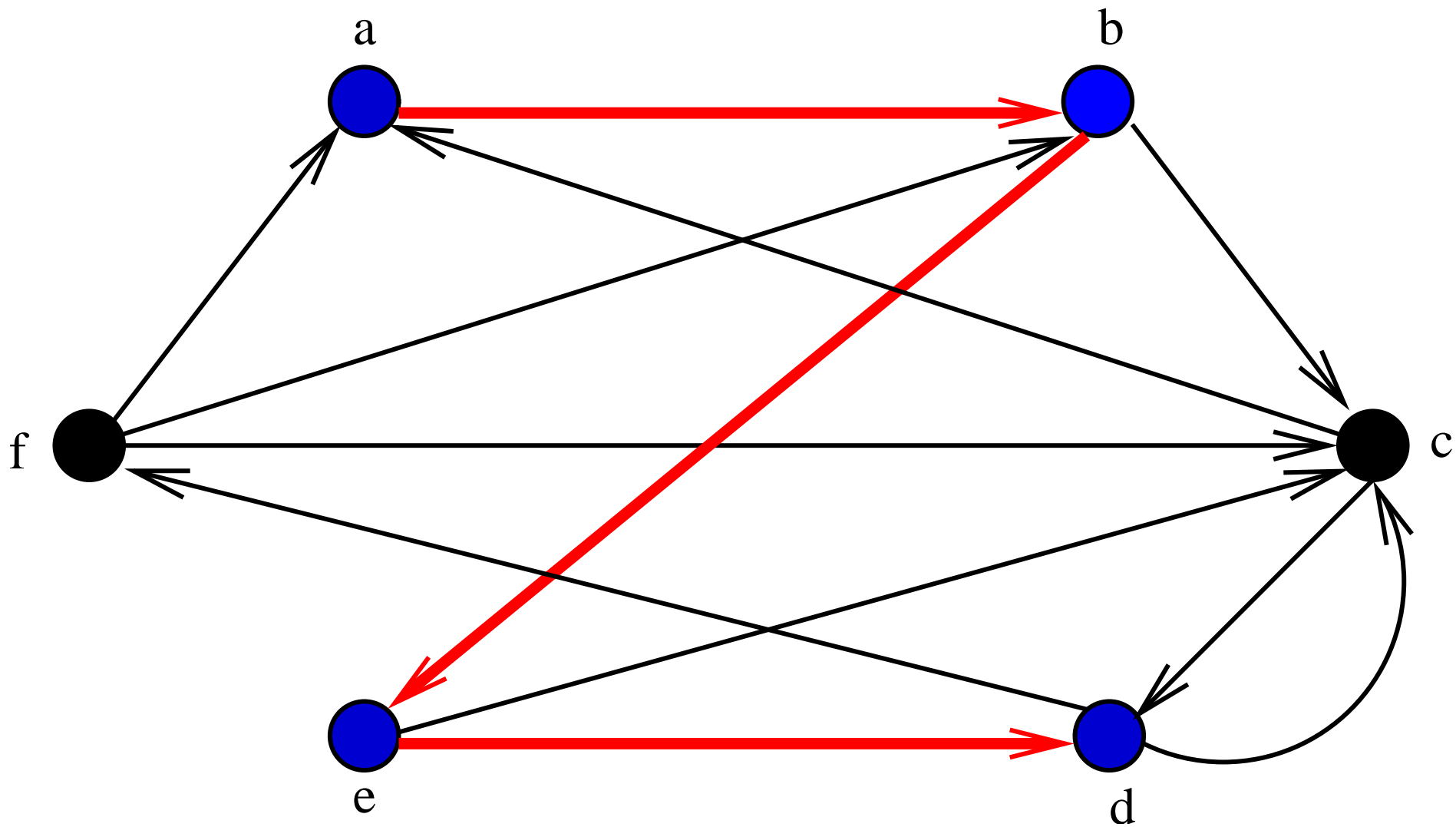


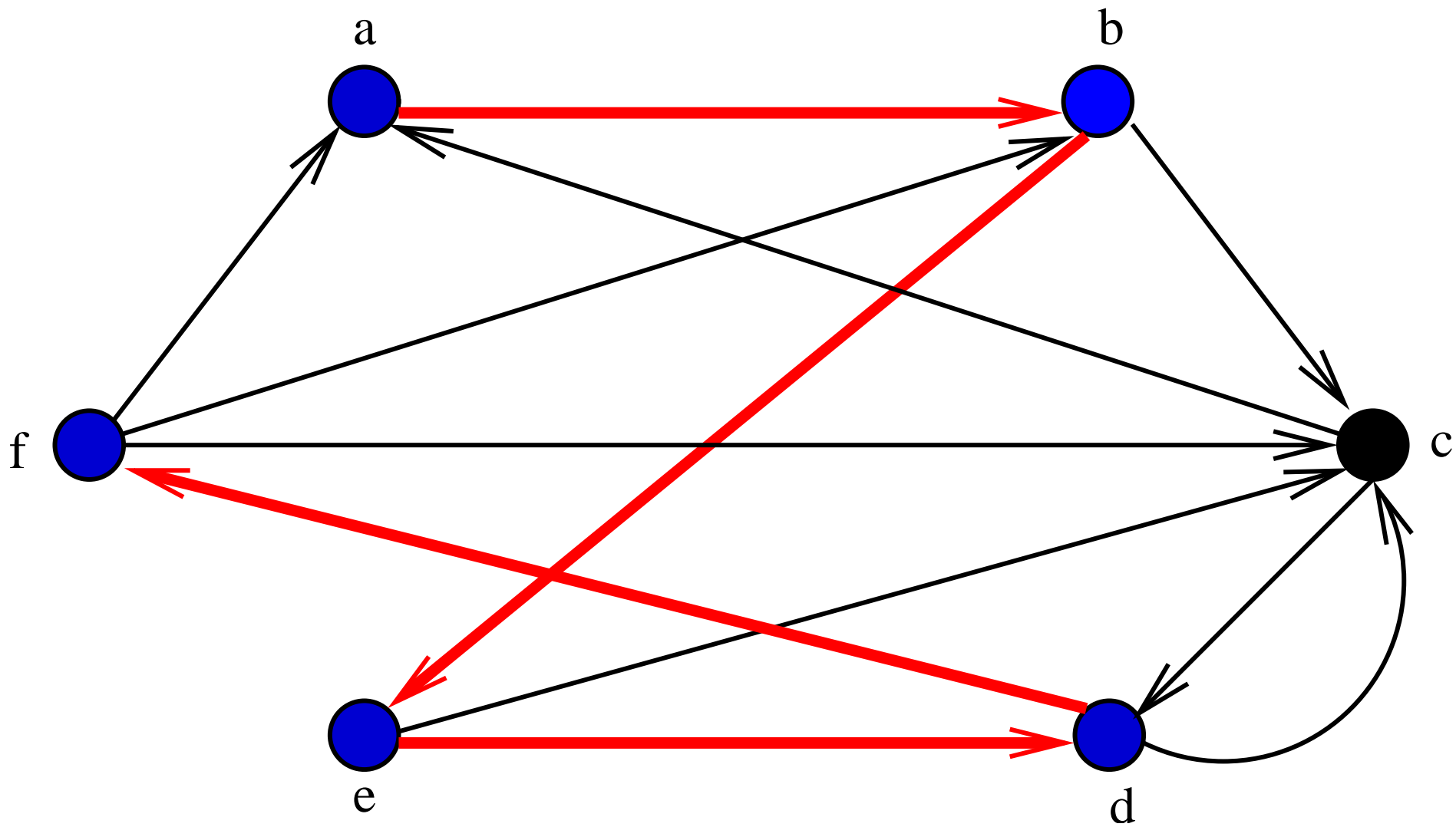


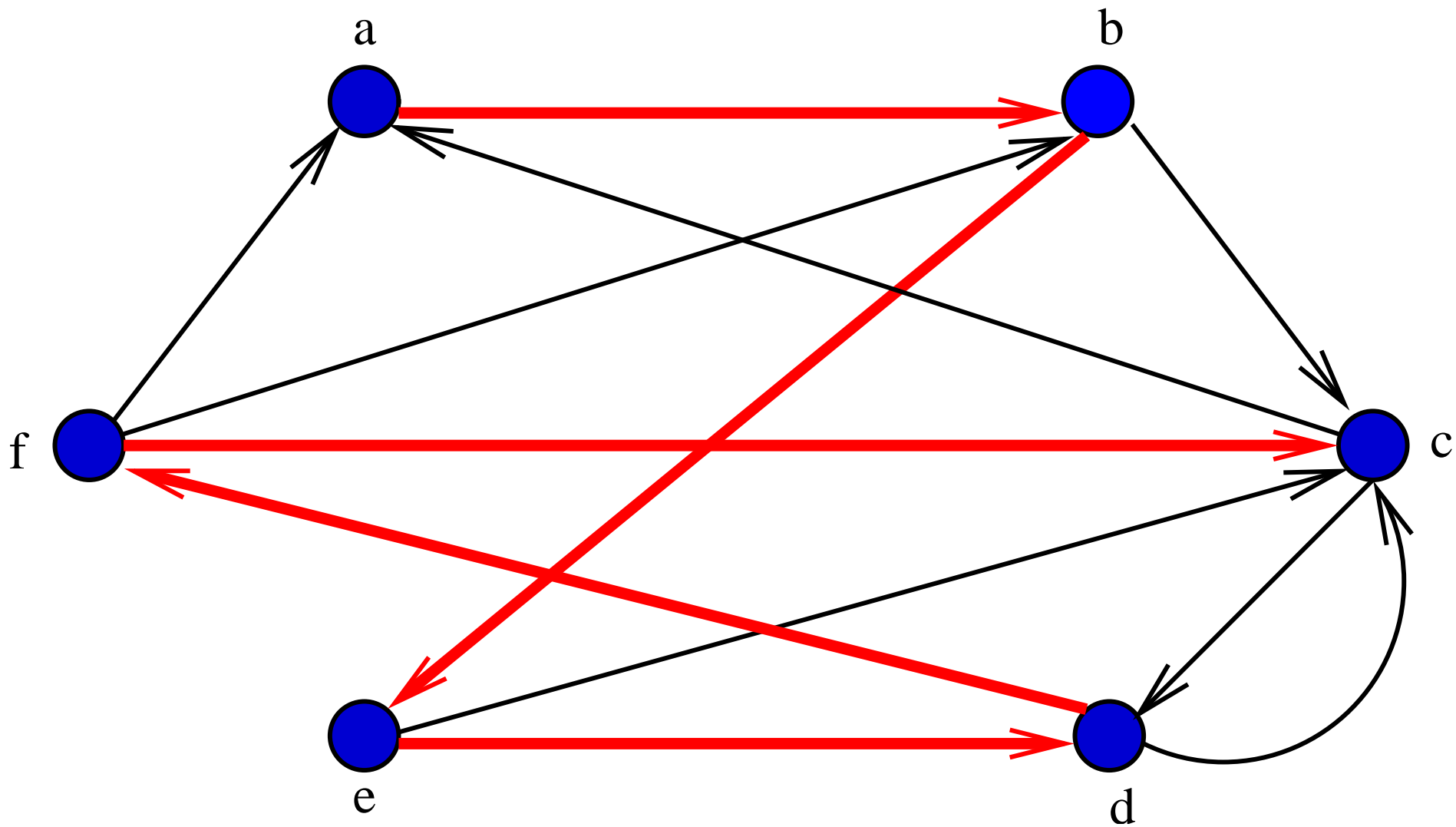


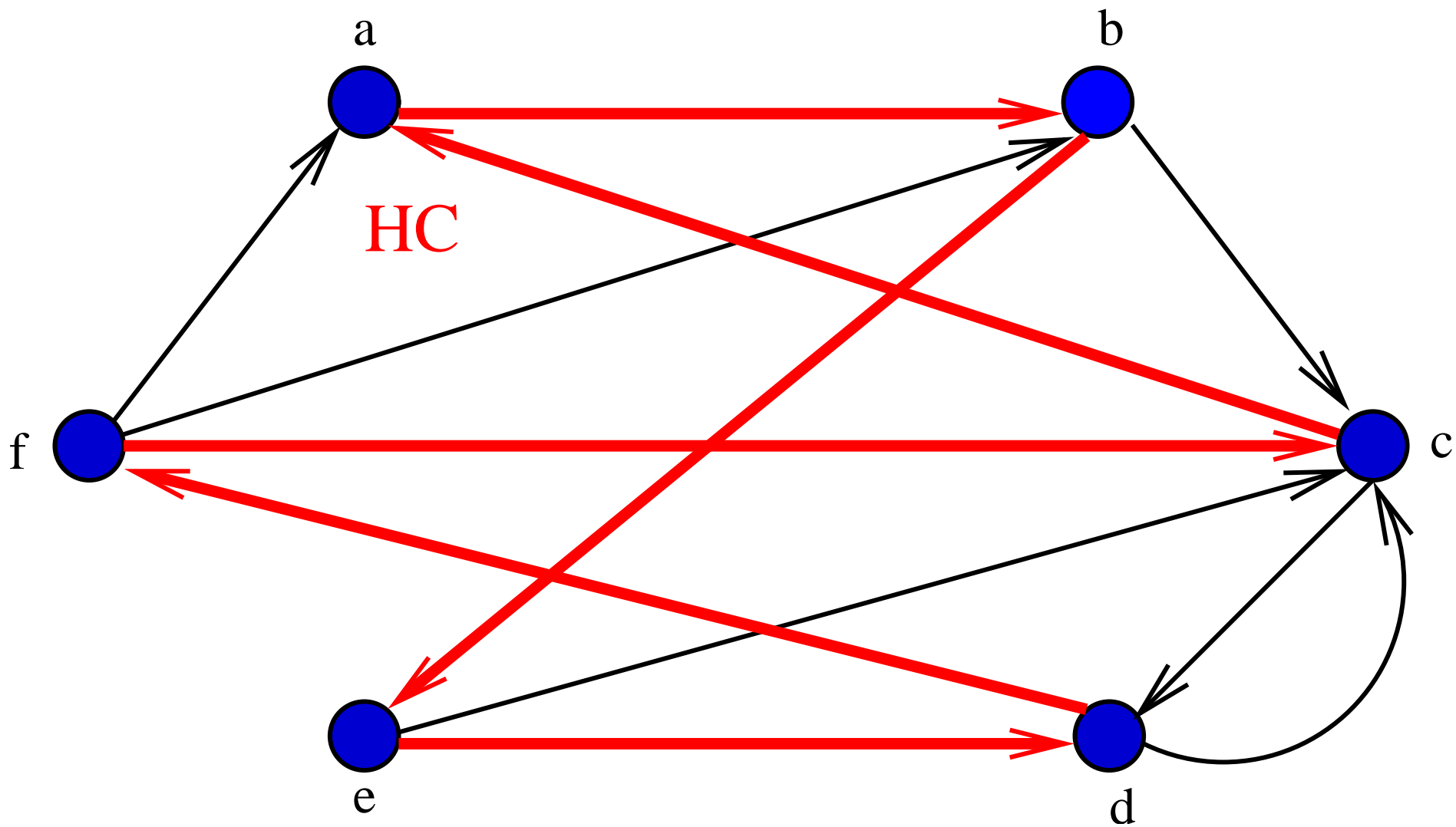


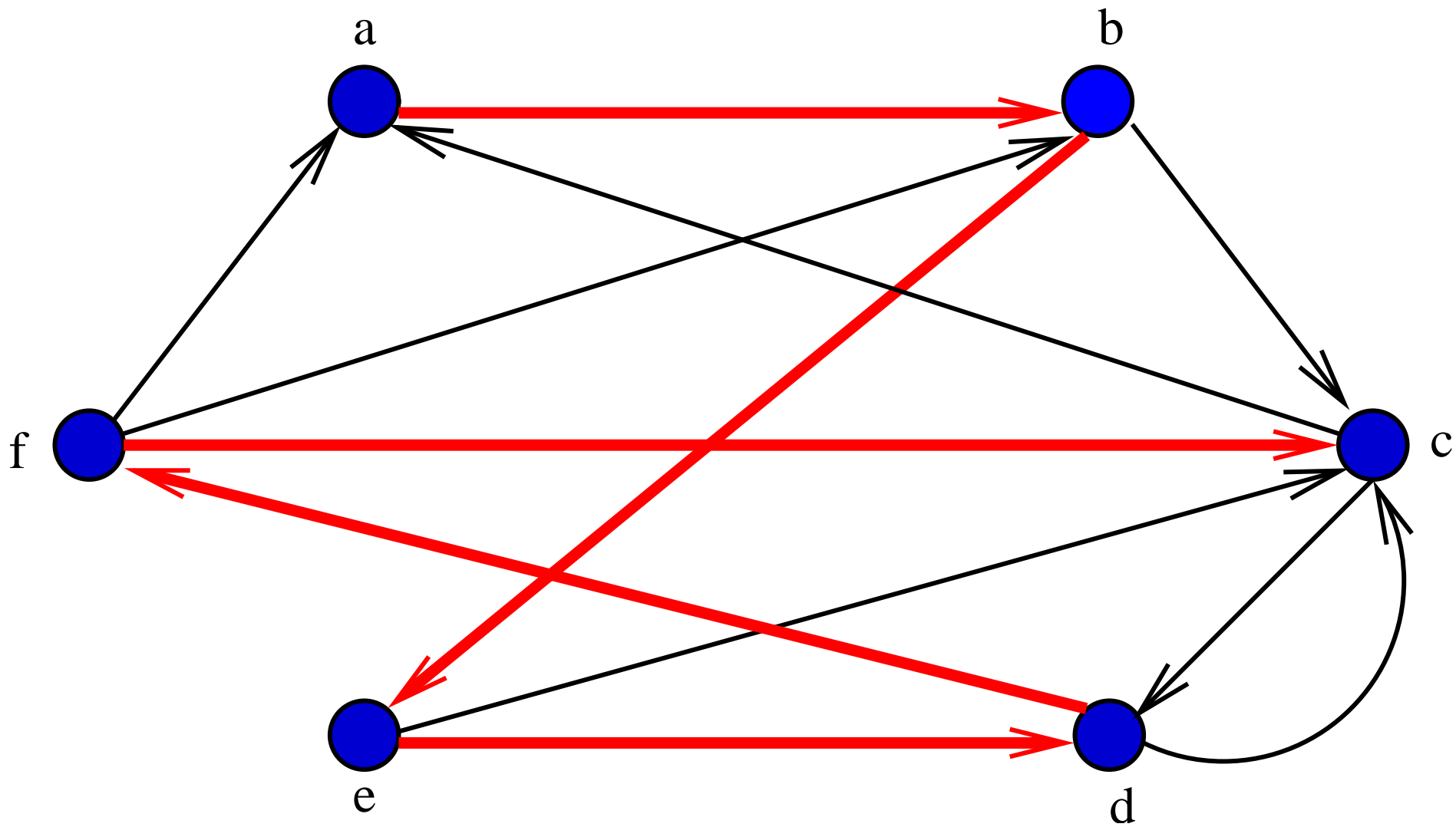


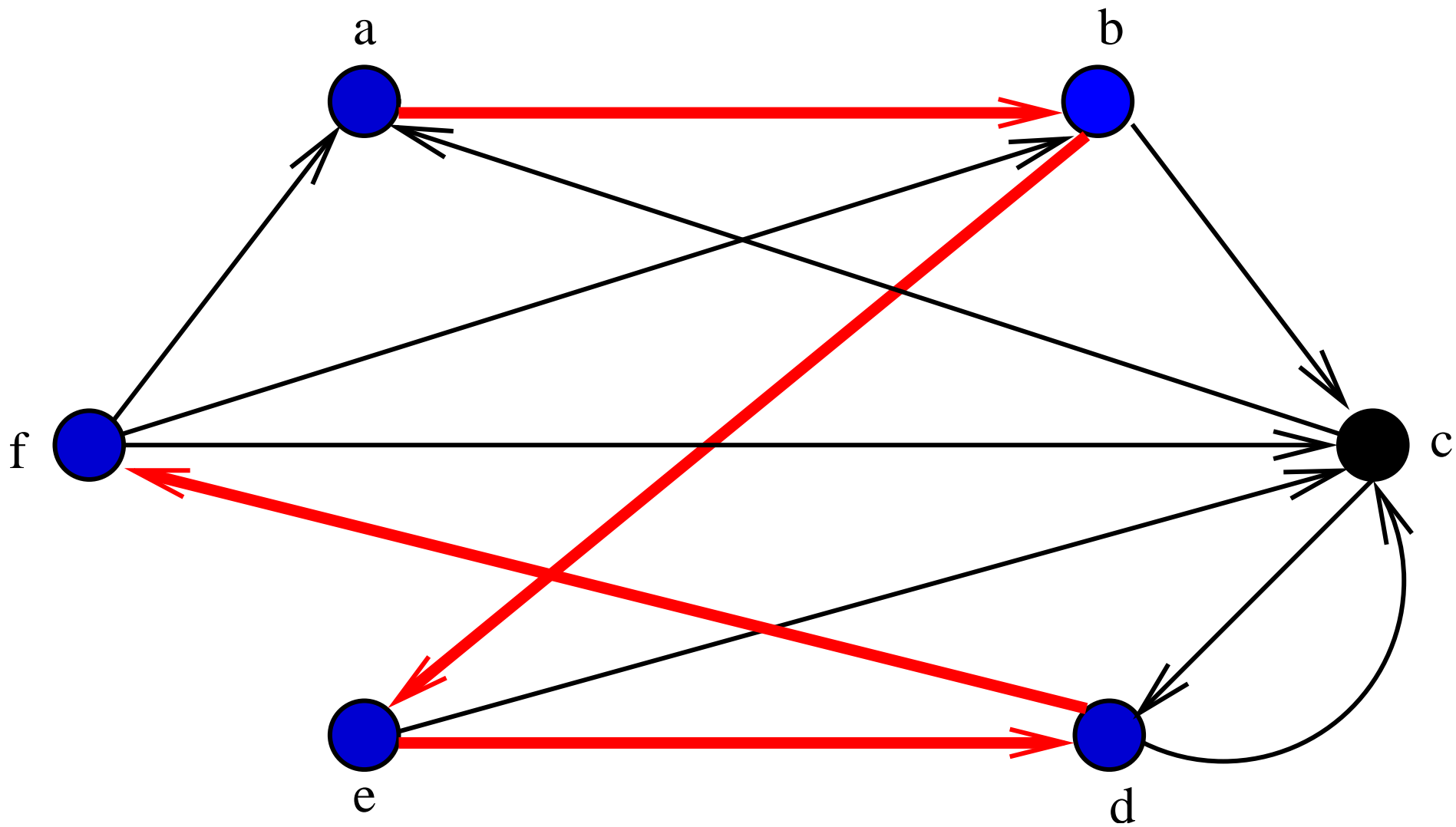


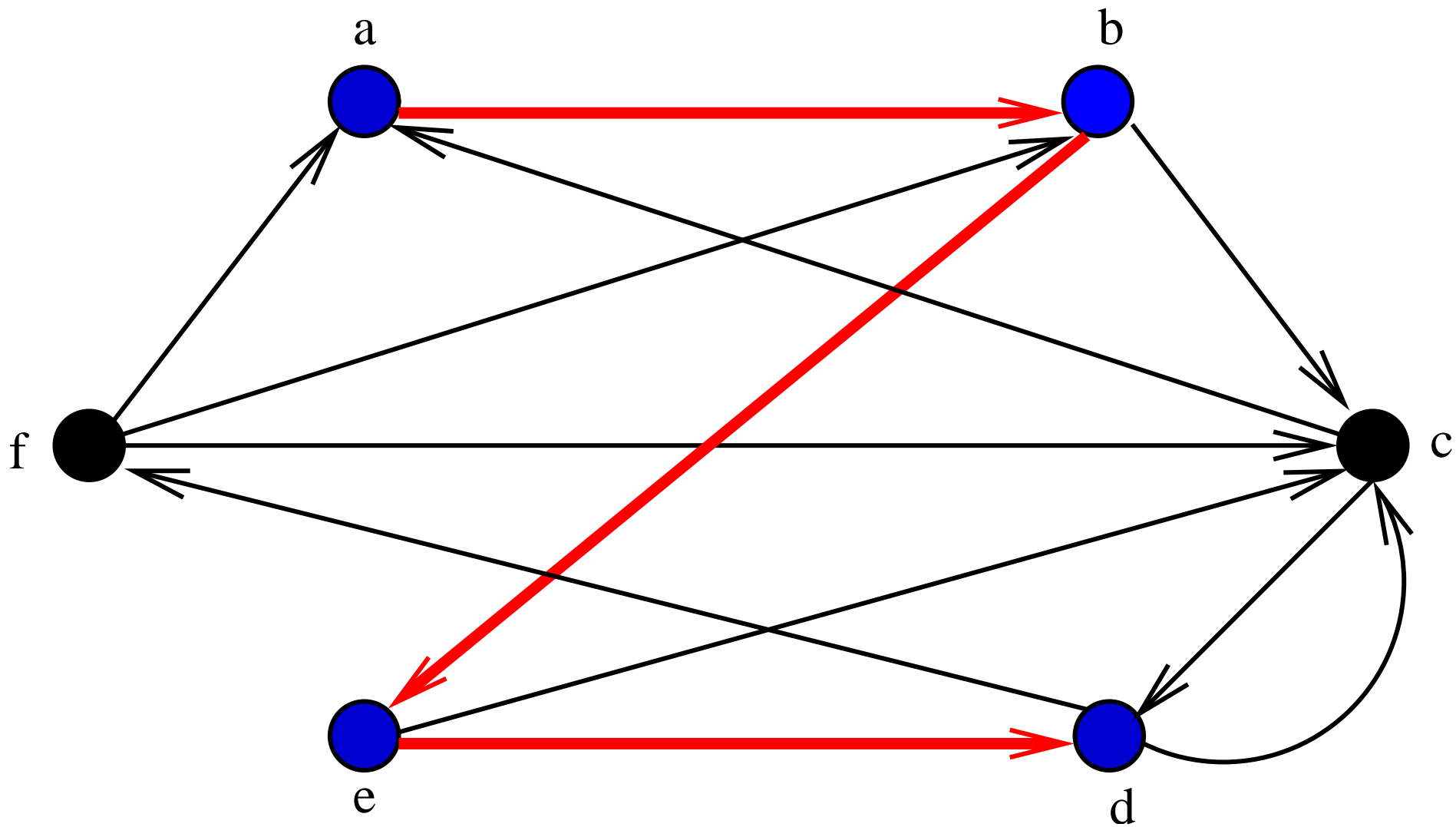


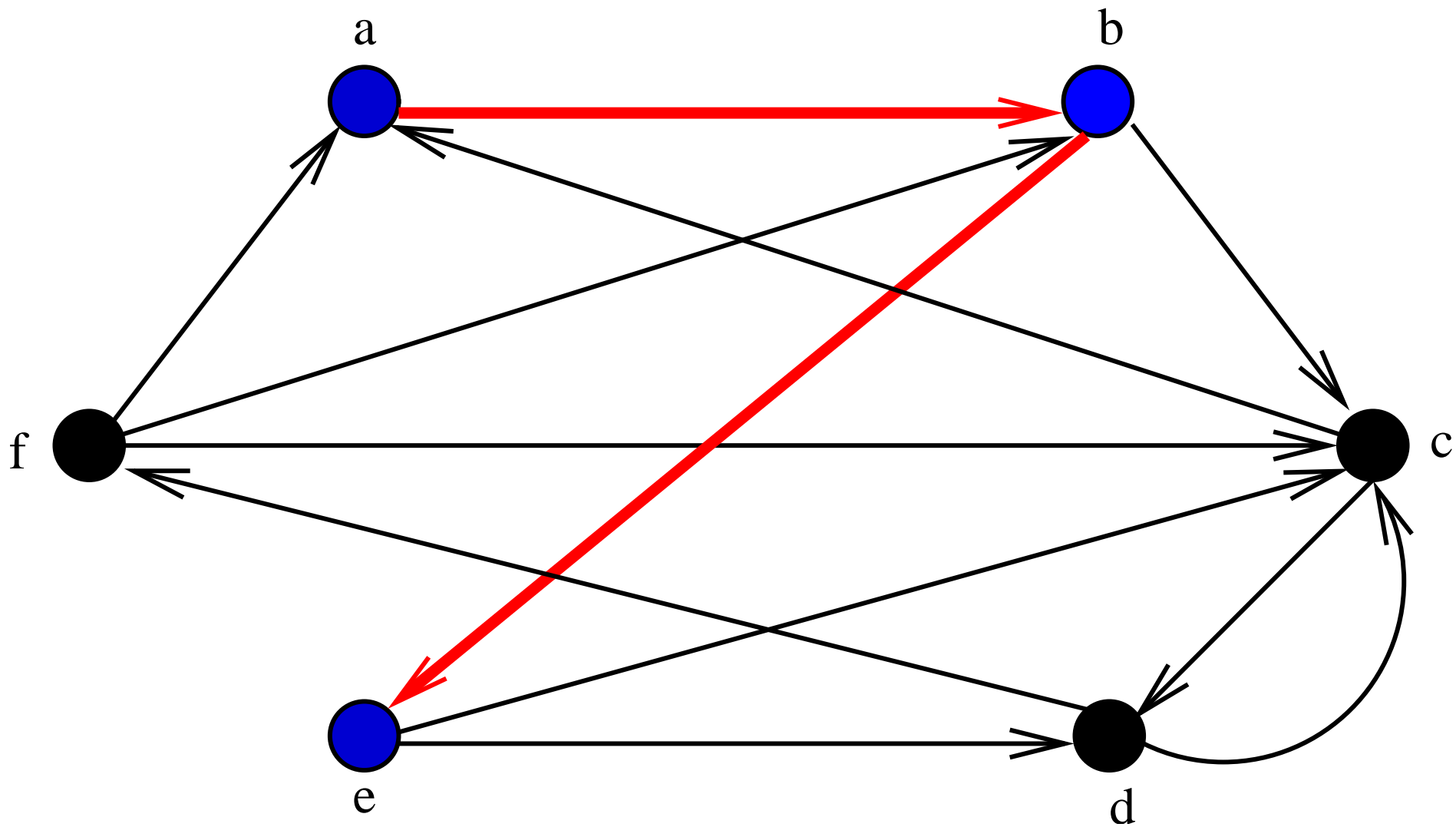


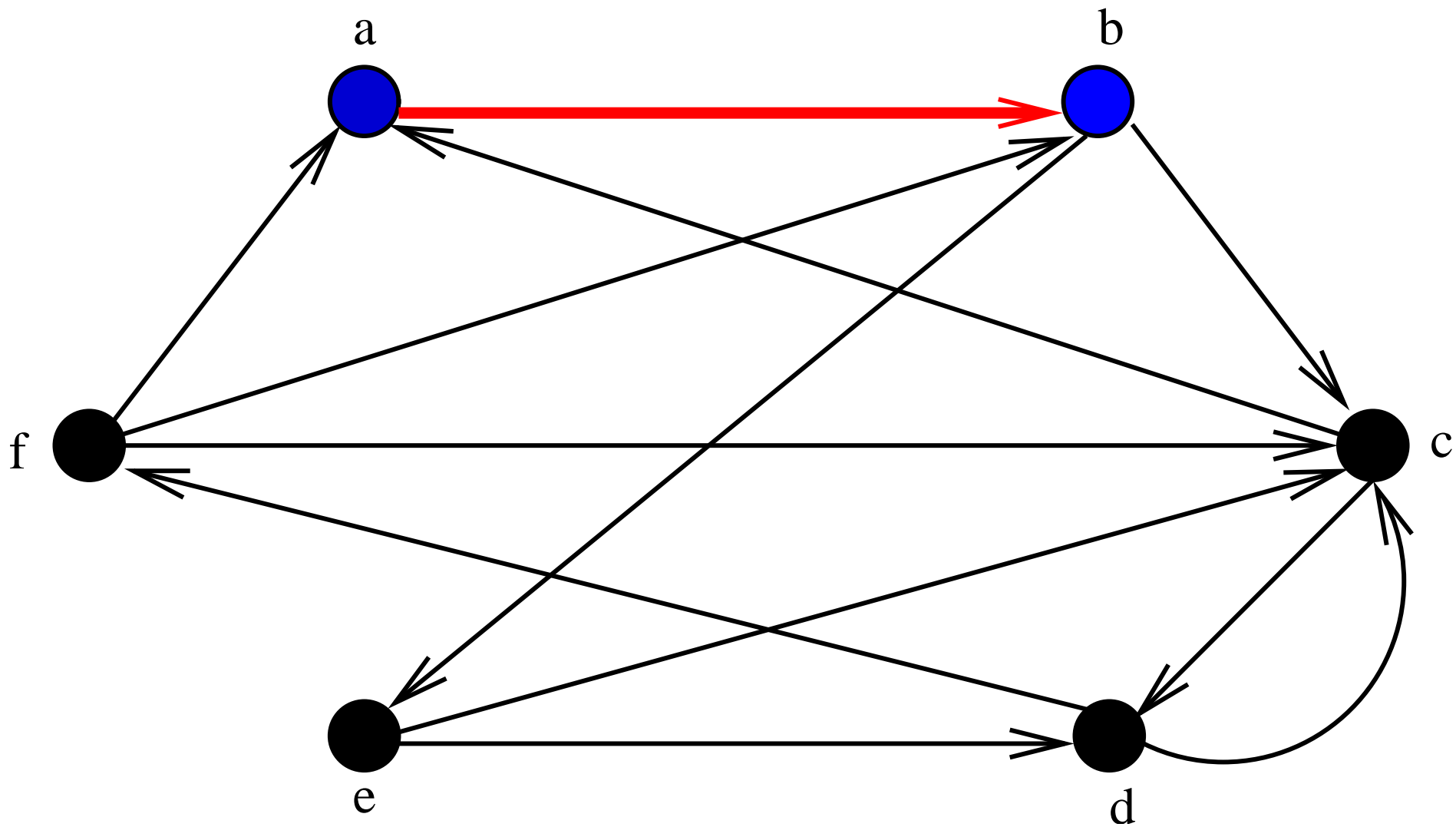


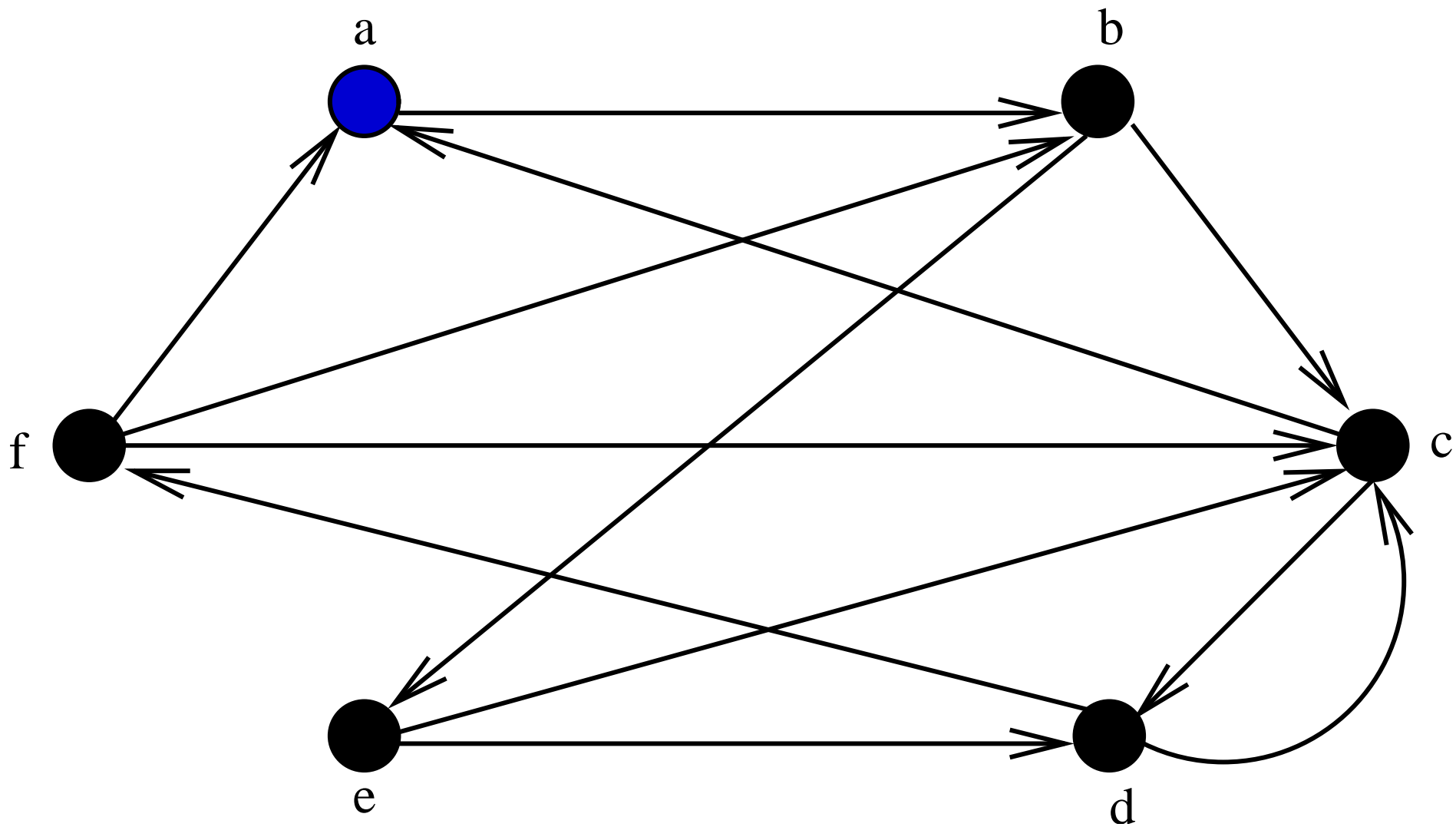


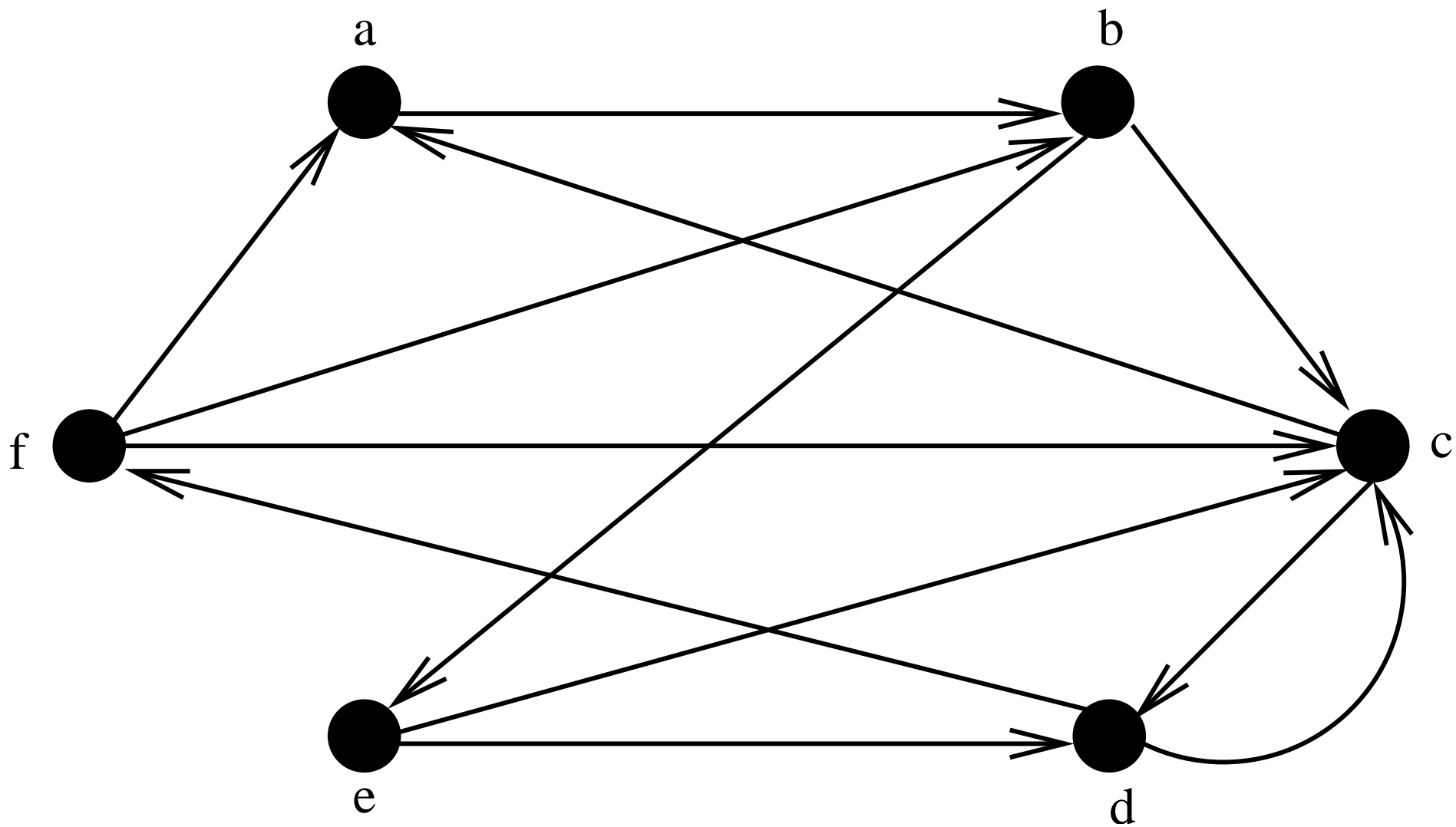






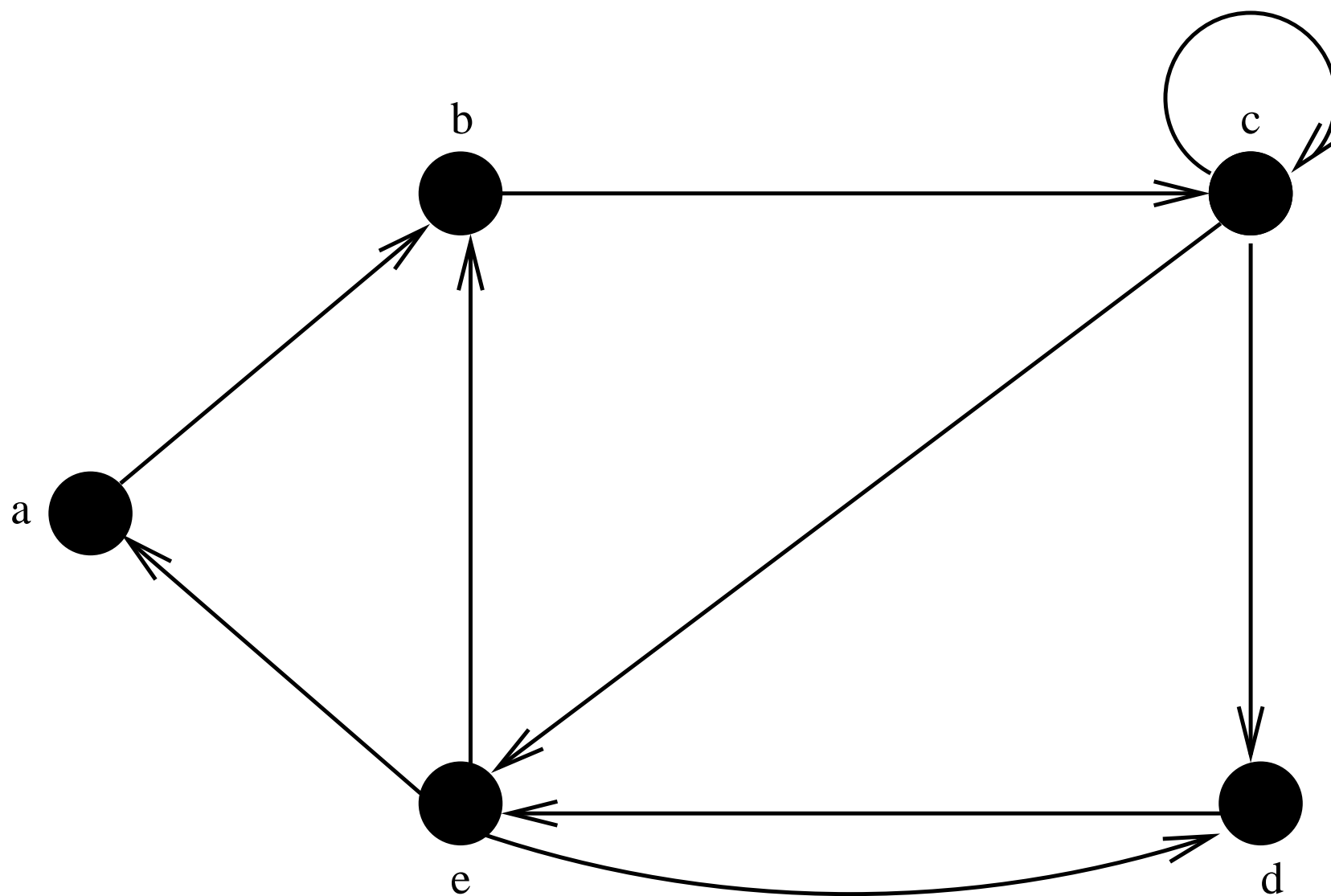






Poszukiwanie cykli Hamiltona za pomocą mnożenia macierzy

- prosta technika bazująca na obserwacji, że (i, j) -ty element k -tej potęgi macierzy przyległości jest liczbą spacerów długości k pomiędzy wierzchołkami i oraz j
- proponowana metoda jest wariacją tego podejścia i prowadzi do uzyskiwania listy ścieżek długości k pomiędzy odpowiednią parą wierzchołków
- w wyniku $n - 1$ kroków procedury otrzymamy listy ścieżek Hamiltona
- ostatni krok to “domknięcie” takich ścieżek i otrzymanie w ten sposób cykli Hamiltona



$$M_1 = \begin{pmatrix} 0 & ab & 0 & 0 & 0 \\ 0 & 0 & bc & 0 & 0 \\ 0 & 0 & 0 & cd & ce \\ 0 & 0 & 0 & 0 & de \\ ea & eb & 0 & ed & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 0 & b & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 \\ 0 & 0 & 0 & d & e \\ 0 & 0 & 0 & 0 & e \\ a & b & 0 & d & 0 \end{pmatrix}$$

Mając dane macierze początkowe możemy, dla $r = 2, \dots, n - 1$, zdefiniować operację “mnożenia” macierzy w następujący sposób:

$$M_r = M_{r-1} * M,$$

gdzie (i, j) -ty element macierzy M_r ma postać:

$$M_r(i, j) = \left\{ \hat{M}_{r-1}(i, k) \circ M(k, j) : 1 \leq k \leq n, \hat{M}_{r-1}(i, k) \in M_{r-1}(i, \right.$$

takie, że ani $\hat{M}_{r-1}(i, k)$ ani $M(k, j)$ nie są równe zeru, ani też nie mają wspólnego symbolu (wierzchołka), natomiast $\hat{M}_{r-1}(i, k) \circ M(k, j)$ oznacza konkatencję odpowiednich ciągów symboli $\hat{M}_{r-1}(i, k)$ oraz $M(k, j)$.

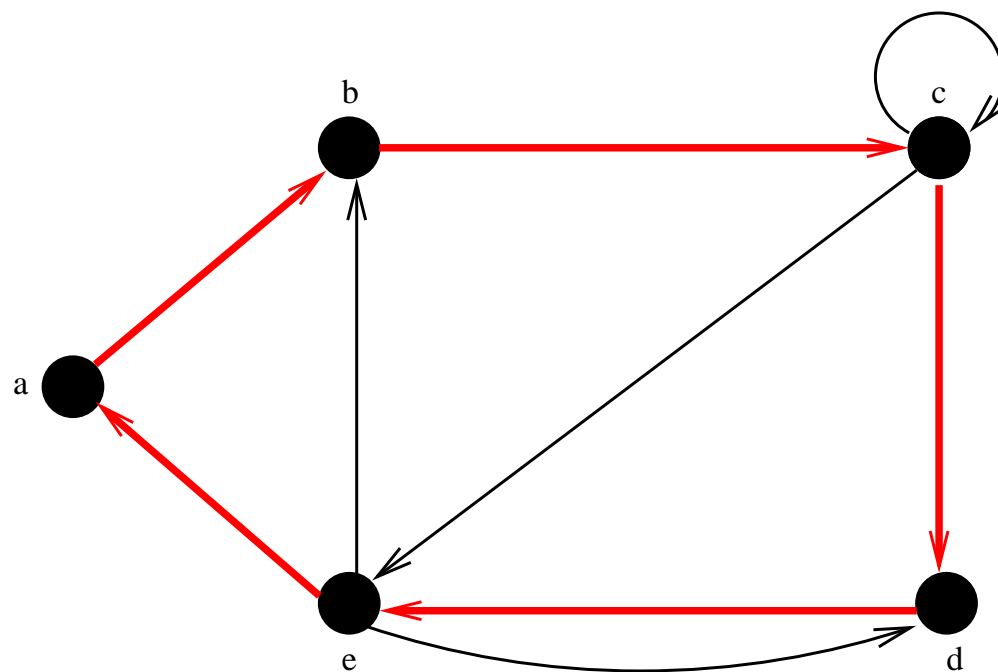
$$M_1 = \begin{pmatrix} 0 & ab & 0 & 0 & 0 \\ 0 & 0 & bc & 0 & 0 \\ 0 & 0 & 0 & cd & ce \\ 0 & 0 & 0 & 0 & de \\ ea & eb & 0 & ed & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 0 & b & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 \\ 0 & 0 & 0 & d & e \\ 0 & 0 & 0 & 0 & e \\ a & b & 0 & d & 0 \end{pmatrix}$$

$$M_2 = \begin{pmatrix} 0 & 0 & abc & 0 & 0 \\ 0 & 0 & 0 & bcd & bce \\ cea & ceb & 0 & ced & cde \\ dea & deb & 0 & 0 & 0 \\ 0 & eab & ebc & 0 & 0 \end{pmatrix}$$

$$M_3 = \begin{pmatrix} 0 & 0 & 0 & abcd & abce \\ bcea & 0 & 0 & bced & bcde \\ cdea & ceab \vee cdeb & 0 & 0 & 0 \\ 0 & deab & debc & 0 & 0 \\ 0 & 0 & eabc & eabd & 0 \end{pmatrix}$$

$$M_3 = \begin{pmatrix} 0 & 0 & 0 & abced & abcde \\ bcdea & 0 & 0 & 0 & 0 \\ 0 & cdeab & 0 & 0 & 0 \\ 0 & 0 & deabc & 0 & 0 \\ 0 & 0 & 0 & eabcd & 0 \end{pmatrix}$$



9.3 Problem Podróżującego Komiwojażera - TSP

- Podróżujący komiwojażer pragnie złożyć wizytę w pewnych miastach i założmy, że chce on powrócić (bądź nie) do punktu startowego. Mając wykaz możliwych połączeń oraz znając czas podróży (koszt podróży) między miastami, jak powinien ułożyć plan podróży aby wizytować miasta dokładnie (co najmniej) jeden raz i aby ta podróż była możliwie najkrótsza (najtańsza)?

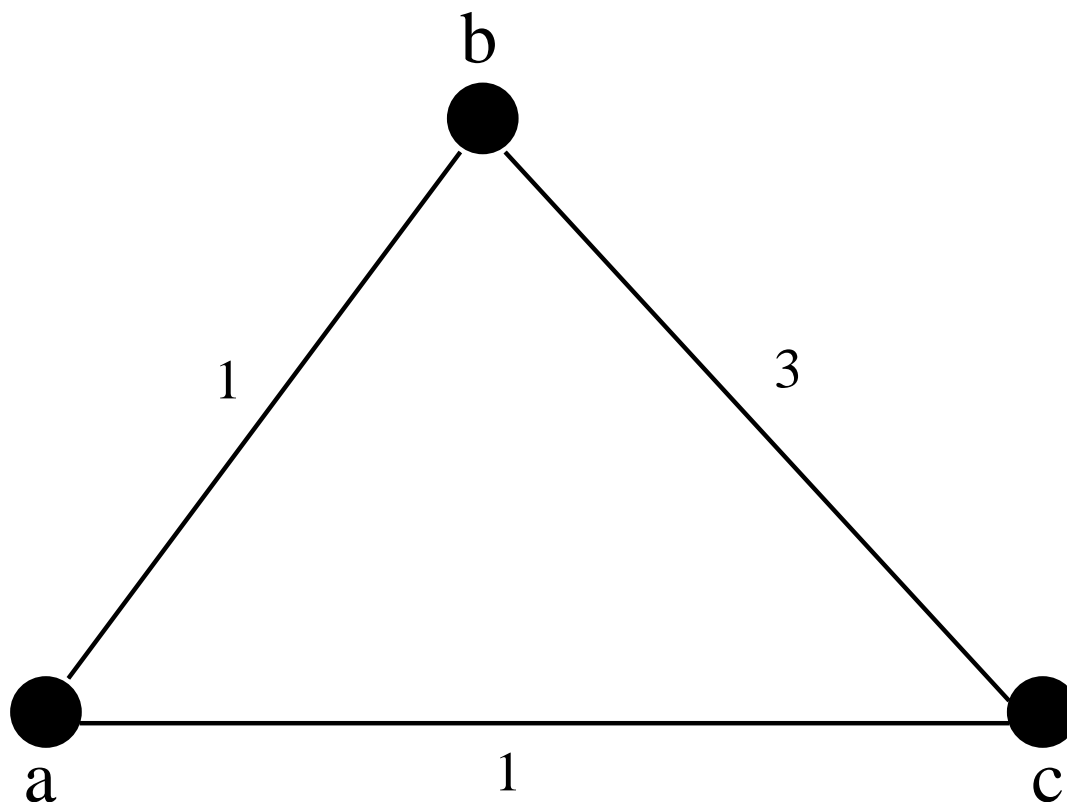
Istnieje wiele różnych definicji problemu TSP przy założeniu, że mamy dany graf G z wagami w na krawędziach. I tak:

- chcemy znaleźć najkrótszy spacer domknięty w G odwiedzający każdy wierzchołek **przynajmniej jeden raz**, lub
- chcemy znaleźć najkrótszy spacer domknięty w G odwiedzający każdy wierzchołek **dokładnie jeden raz**, to znaczy znaleźć cykl Hamiltona o najmniejszej wadze.

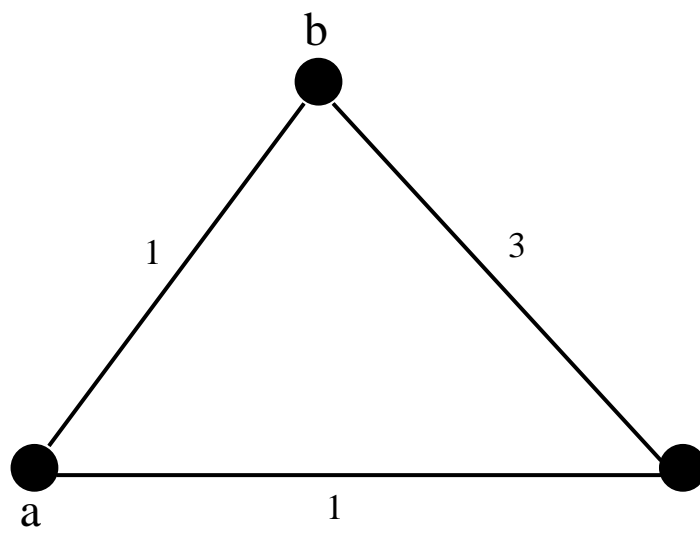
- Przyjmijmy ogólniejszą definicję problemu TSP, to znaczy zakładamy, że szukamy najkrótszego spaceru domkniętego w G odwiedzający każdy wierzchołek **przynajmniej jeden raz**.
- w grafie hamiltonowskim rozwiązanie problemu TSP nie jest równoważne ze znalezieniem cyklu Hamiltona o najmniejszej wadze - ma to miejsce w przypadku gdy wagi grafu nie jest spełniają nierówności trójkąta, to znaczy nie dla każdej pary wierzchołków u oraz v spełniona jest nierówność

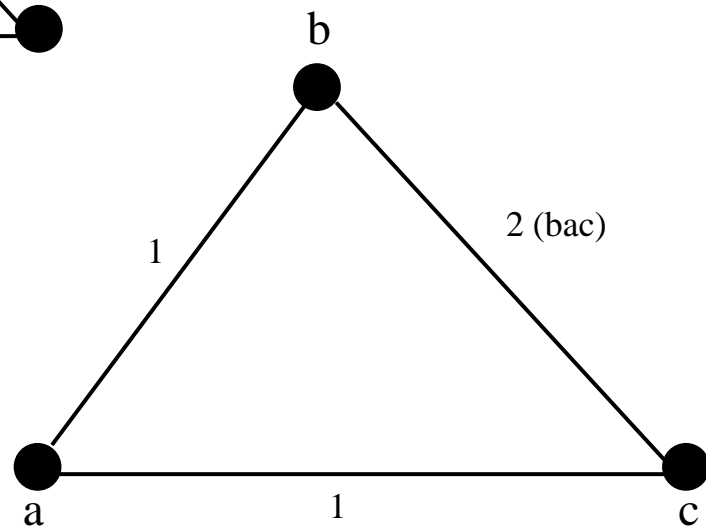
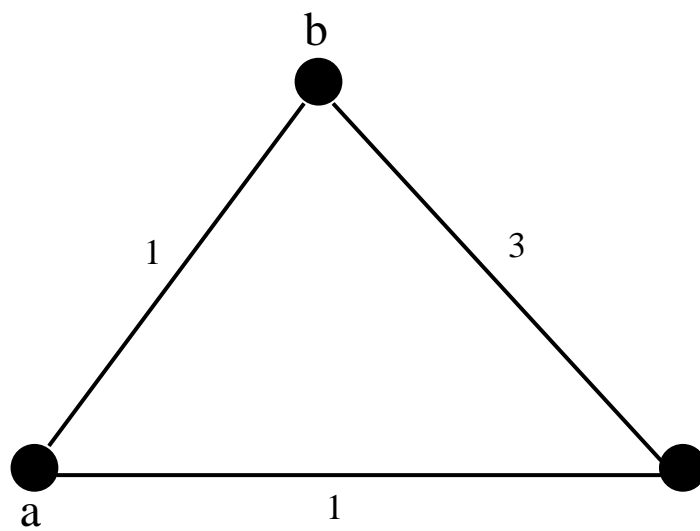
$$w(u, v) \leq w(u, x) + w(x, v)$$

(dla wszystkich wierzchołków $x \neq u, v$).



- można pokazać równoważność problemu TSP dla ważonego grafu $G = (V, E)$ z problemem znalezienia cyklu Hamiltona o najmniejszej wadze w grafie pełnym $G' = (V', E')$ gdzie $V' = V$, natomiast każda krawędź $e = (u, v) \in E'$ ma wagę $w(u, v)$ równą wadze najkrótszej ścieżki pomiędzy u oraz v
- zauważmy, że wtedy każda krawędź z E' odpowiada ścieżce złożonej z jednej lub więcej krawędzi grafu G (dlatego wygodnie jest znakować krawędzie grafu G' odpowiednimi ścieżkami z G)
- jeszcze raz podkreślmy, że gdy wagi w G nie spełniają nierówności trójkąta to rozwiązanie TSP oznacza znalezienie w G najkrótszego spaceru w którym każdy wierzchołek odwiedzamy co najmniej jeden raz!





Twierdzenie . Rozwiązanie problemu TSP w grafie G odpowiada i jest tej samej długości co najkrótszy cykl Hamiltona w grafie pełnym G' .

- w świetle powyższego twierdzenia możemy w dalszych rozważaniach przyjąć, że poszukujemy najkrótszego cyklu Hamiltona w ważonym grafie pełnym
- przeszukiwanie **wszystkich** cykli Hamiltona w grafie pełnym to zły pomysł nawet dla niezbyt dużych grafów, ponieważ takich cykli jest $\frac{1}{2}(n-1)!$ - na przykład, przy założeniu, że komputer dokonuje 10^8 operacji dodawania na sekundę, to znalezienie wszystkich cykli Hamiltona w grafie pełnym na n wierzchołkach zabrało by dla $n = 15$ ok. 3 godziny, dla $n = 20$ ok. 800 lat a dla $n = 50$ ok. 10^{49} lat!

W poszukiwaniu bardziej efektywnych rozwiązań problemu TSP skoncentrujemy się najpierw na następujących, jak się dalej okaże, równoważnych problemach:

- A. Znalezienie najkrótszej ścieżki Hamiltona w grafie z wagami.
- B. Znalezienie najkrótszej ścieżki Hamiltona między ustalonymi wierzchołkami w grafie z wagami.

- Zaobserwujemy najpierw istnienie oczywistego związku pomiędzy problemem **TSP** a problemem **MST**:

- *ponieważ ścieżka Hamiltona jest rozpiętym drzewem to rozwiązanie problemu **TSP** sprowadza się do rozwiązania problemu **MST**, przy nałożonym dodatkowo ograniczeniu na stopnie wierzchołków rozpiętego drzewa.*
- *w ogólności, za miarę “bliskości” rozpiętego drzewa T i ścieżki Hamiltona, można przyjąć wyrażenie*

$$\epsilon_T = \sum_{d_i^T > 2} (d_i^T - 2) = \sum_{i=1}^n |d_i^T - 2| - 2,$$

gdzie d_i^T to stopień i -tego wierzchołka drzewa T .

- Można zatem przeformułować obydwa wymienione powyżej problemy w następujący sposób:

- A. Najkrótsza ścieżka Hamiltona.** Znajdź minimalne rozpięte drzewo w grafie G takie, że stopnie wierzchołków tego drzewa nie przekraczają 2.
- B. Najkrótsza ścieżka Hamiltona między ustalonymi wierzchołkami.** Dane są dwa wierzchołki $v_1, v_2 \in V$. Znajdź minimalne rozpięte drzewo w grafie G takie, że wierzchołki v_1 i v_2 mają stopień 1, natomiast stopnie pozostałych wierzchołków drzewa nie przekraczają 2.

Twierdzenie . Niech graf G dany będzie macierzą wag $W = (w_{ij})$, a graf G^* macierzą wag $W^* = (w_{ij}^*)$, gdzie

$$w_{ij}^* = \begin{cases} w_{ij} + 2K & \text{dla } i = 1, j = 2 \text{ lub } i = 2, j = 1, \\ w_{ij} + K & \text{dla } i \leq 2, j > 2 \text{ lub } i > 2, j \leq 2, \\ w_{ij} & \text{dla pozostałych } i, j. \end{cases}$$

Wówczas dla dostatecznie dużej stałej K (na przykład dla $K \geq |V(G)| \max\{w_{ij} : w_{ij} < \infty\}$) rozwiązanie zadania **A** (znalezienie najkrótszej ścieżki Hamiltona) o wadze mniejszej od $3K$ dla grafu G^* jest rozwiązaniem zadania **B** (znalezienie najkrótszej ścieżki Hamiltona między wierzchołkami v_1 i v_2) dla grafu G . Jeżeli natomiast najkrótsza ścieżka Hamiltona w grafie G^* ma wagę większą od $3K$, to w grafie G ścieżka Hamiltona pomiędzy v_1 i v_2 nie jest najkrótsza.

Dowód. Dowolną ścieżkę Hamiltona można zaliczyć do jednej z trzech kategorii: (1) żaden z wierzchołków v_1, v_2 nie jest końcem tej ścieżki, (2) jeden z wierzchołków v_1, v_2 jest końcem tej ścieżki, (3) wierzchołki v_1, v_2 są końcami tej ścieżki.

Waga dowolnej ścieżki Hamiltona w grafie a macierzą wag W^* przekracza wagę tej ścieżki w grafie z macierzą W , odpowiednio o: $4k$, jeżeli ścieżka jest typu (1), $3k$, jeżeli ścieżka jest typu (2), $2k$, jeżeli ścieżka jest typu (3).

Ponadto, ponieważ K przewyższa wagę najdłuższej ścieżki Hamiltona, to waga (w grafie z W^*) najdłuższej ścieżki Hamiltona typu (3) jest mniejsza niż waga najdłuższej ścieżki Hamiltona typu (2), a waga najdłuższej ścieżki Hamiltona typu (2) jest mniejsza niż waga najdłuższej ścieżki Hamiltona typu (1). Zatem rozwiązaniem zadania A z macierzą wag W^* może być jedynie najkrótsza ścieżka Hamiltona typu (3), a to oznacza, że jest ona rozwiązaniem zadania B z macierzą wag W . ■

Twierdzenie . Niech graf G dany będzie macierzą wag $W = (w_{ij})$. Utwórzmy z grafu G nowy graf G^{**} poprzez dodanie dwóch nowych wierzchołków u i v połączonych ze wszystkimi wierzchołkami grafu G krawędziami o identycznej wadze w . Wówczas rozwiązanie zadania B (znalezienie najkrótszej ścieżki Hamiltona między wierzchołkami u i v) dla grafu G^{**} jest rozwiązaniem zadania A dla grafu G . ■

Twierdzenie . Niech graf G dany będzie macierzą wag $W = (w_{ij})$, a graf G^* macierzą wag $W^* = (w_{ij}^*)$, gdzie $w_{ij}^* = w_{ij} + p(i) + p(j)$, przy czym $p(\cdot)$ jest dowolnym wektorem $|V(G)|$ -wymiarowym liczb rzeczywistych. Wówczas rozwiązanie zadania B dla grafu G^* jednoznacznie wyznacza rozwiązanie zadania B (dla tych samych wierzchołków końcowych ścieżki Hamiltona) dla grafu G .

Dowód. Niech F_W oznacza wagę ścieżki Hamiltona pomiędzy wierzchołkami v_1, v_2 w grafie z macierzą wag W .

Ponieważ w ścieżce dowolny wierzchołek, poza końcowymi, jest przyległy do dokładnie dwóch wierzchołków, dlatego waga F_{W^*} tej ścieżki w grafie z wagami W^* różni się od F_W o wielkość

$$F_{W^*} - F_W = p(1) + p(2) + 2 \sum_{j=3}^n p(j).$$

Zauważmy, że wielkość ta nie zależy od wyboru ścieżki Hamiltona, co kończy dowód. 

Problem Podróżującego Komiwojażera – rozwiązanie zadania **A** (Metoda przeszukiwania wszystkich przypadków)

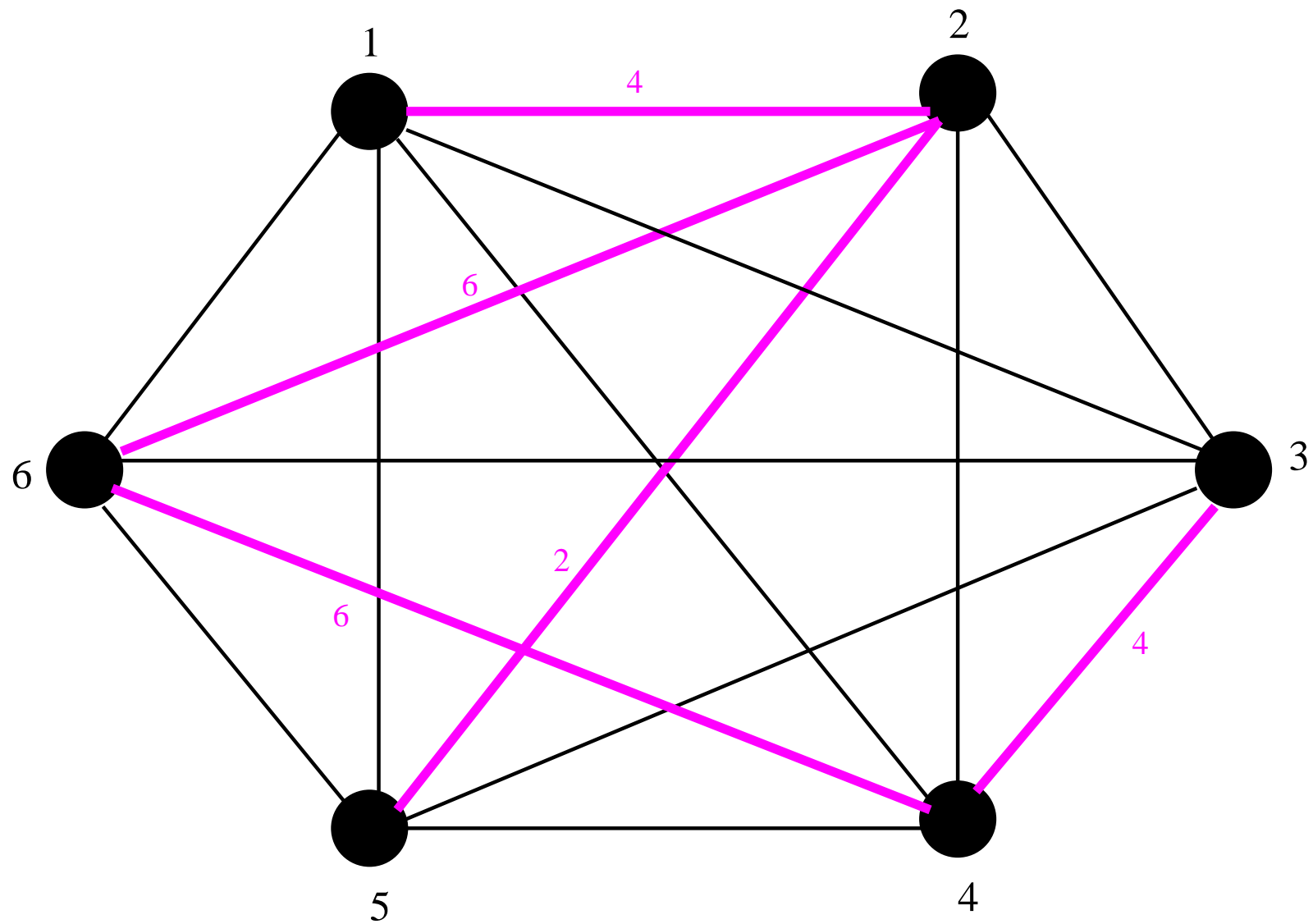
1. Znajdujemy minimalne drzewo rozpięte T grafu G o macierzy wag W .
2. Jeżeli nie istnieje wierzchołek stopnia > 2 , to STOP – znalezione drzewo jest szukaną najkrótszą ścieżką H . Jeżeli v jest taki, że $d(v) > 2$ i e_1, e_2, \dots, e_k są krawędziami wychodzącymi z v , to przechodzimy do rozpatrywania naszego zadania dla grafów:

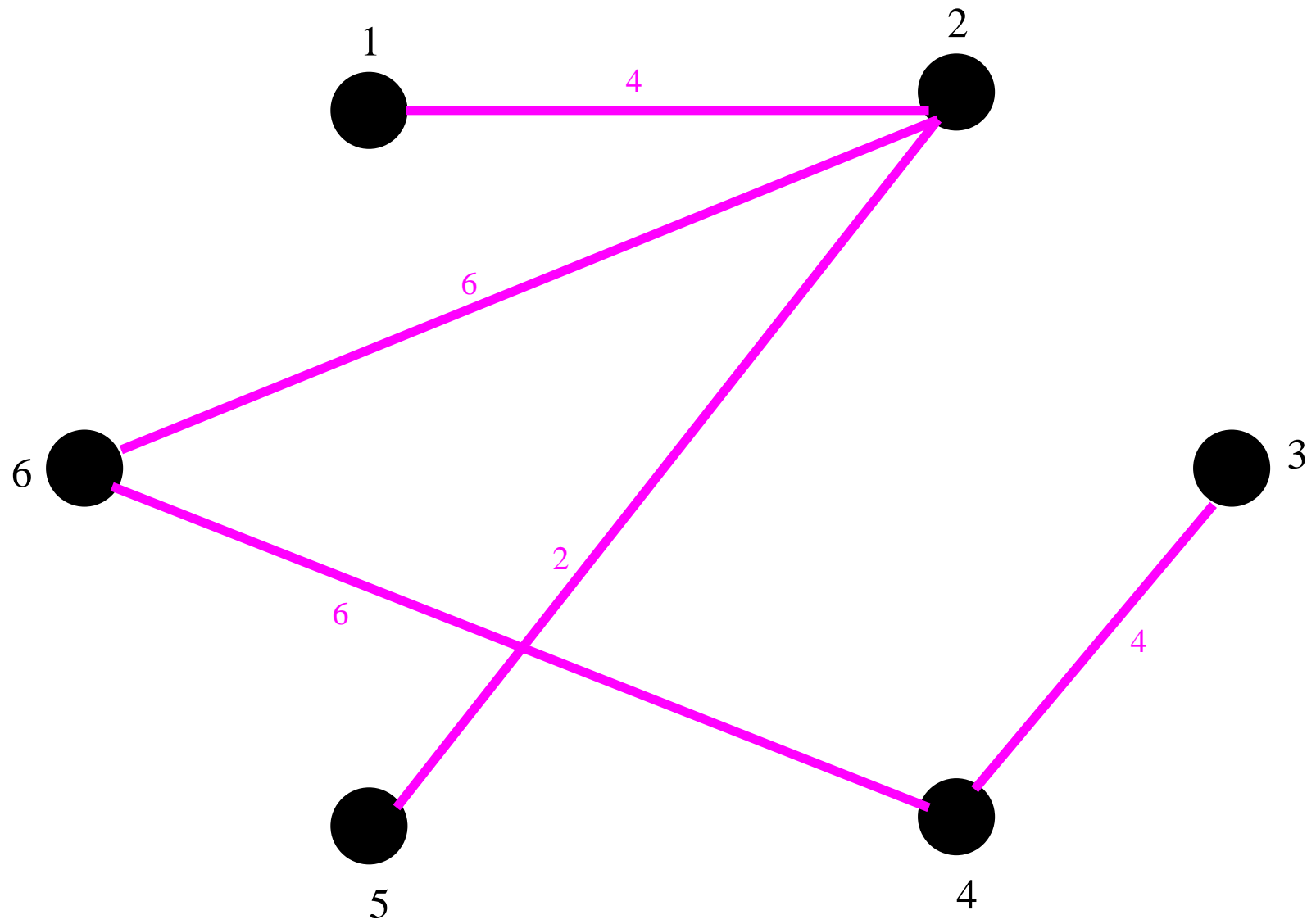
$$G_1, G_2, \dots, G_k,$$

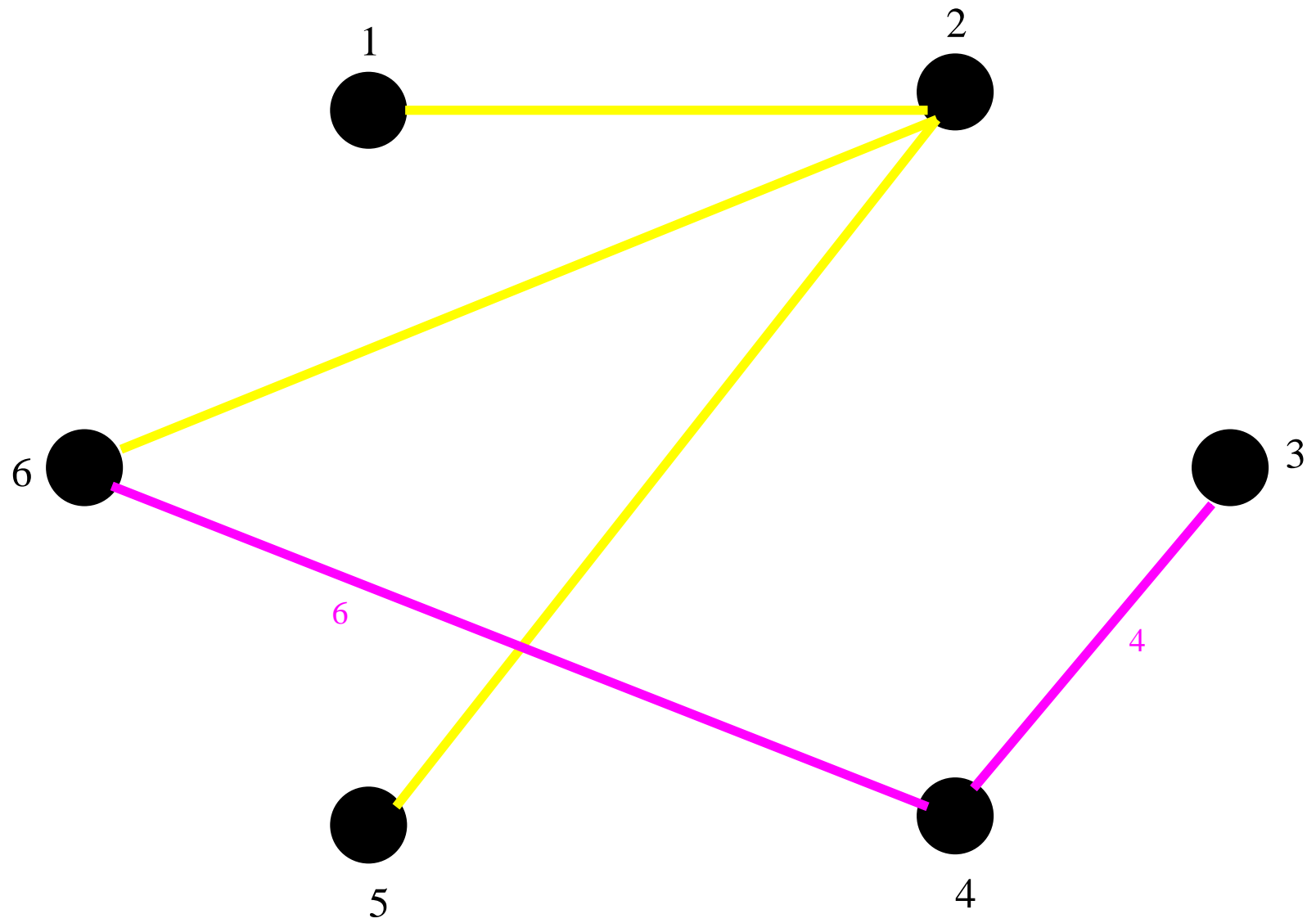
gdzie macierz wag grafu G_i powstaje z macierzy wag W poprzez zastąpienie wagi krawędzi e_i nieskończonością, $i = 1, 2, \dots, k$.
(Zauważmy, że żadna iteracja nie może zmniejszyć wagi rozpiętego drzewa)

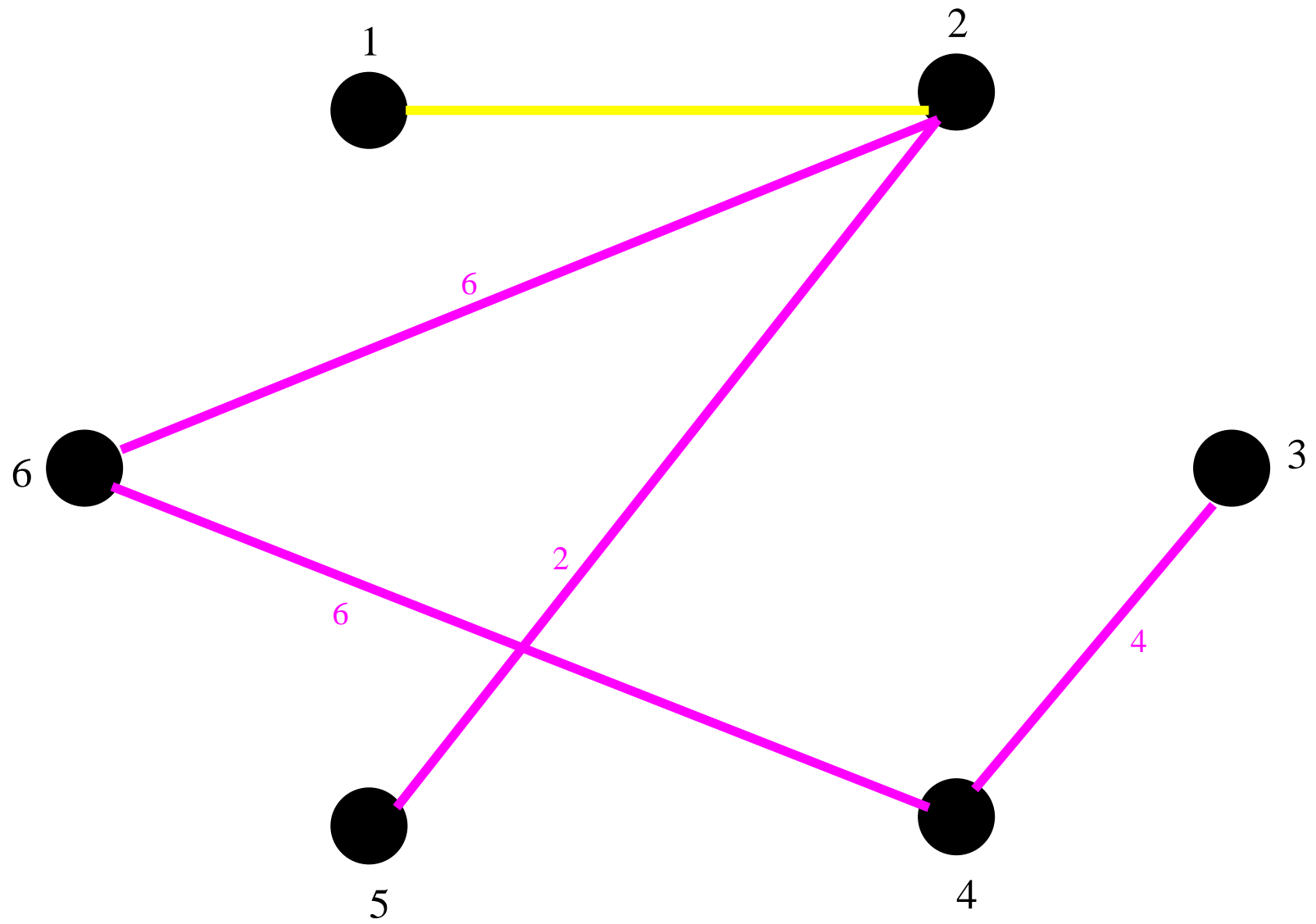
Przykład . Wyznaczyć za pomocą algorytmu przeszukiwania wszystkich przypadków najkrótszą ścieżkę Hamiltona w grafie o macierzy wag:

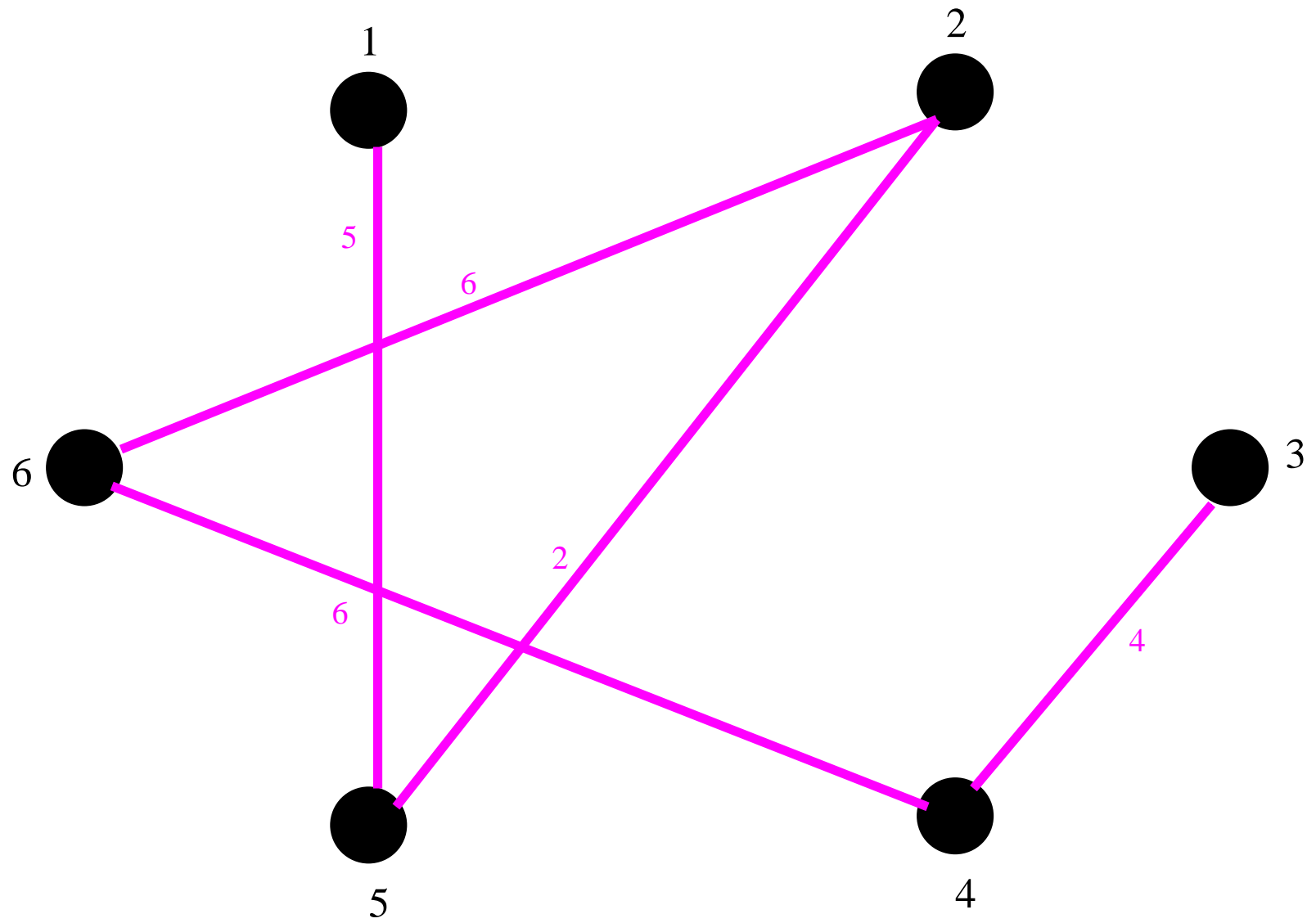
$$W = \begin{pmatrix} \infty & 4 & 10 & 18 & 5 & 10 \\ 4 & \infty & 12 & 8 & 2 & 6 \\ 10 & 12 & \infty & 4 & 18 & 16 \\ 18 & 8 & 4 & \infty & 14 & 6 \\ 5 & 2 & 18 & 14 & \infty & 16 \\ 10 & 6 & 16 & 6 & 16 & \infty \end{pmatrix} .$$

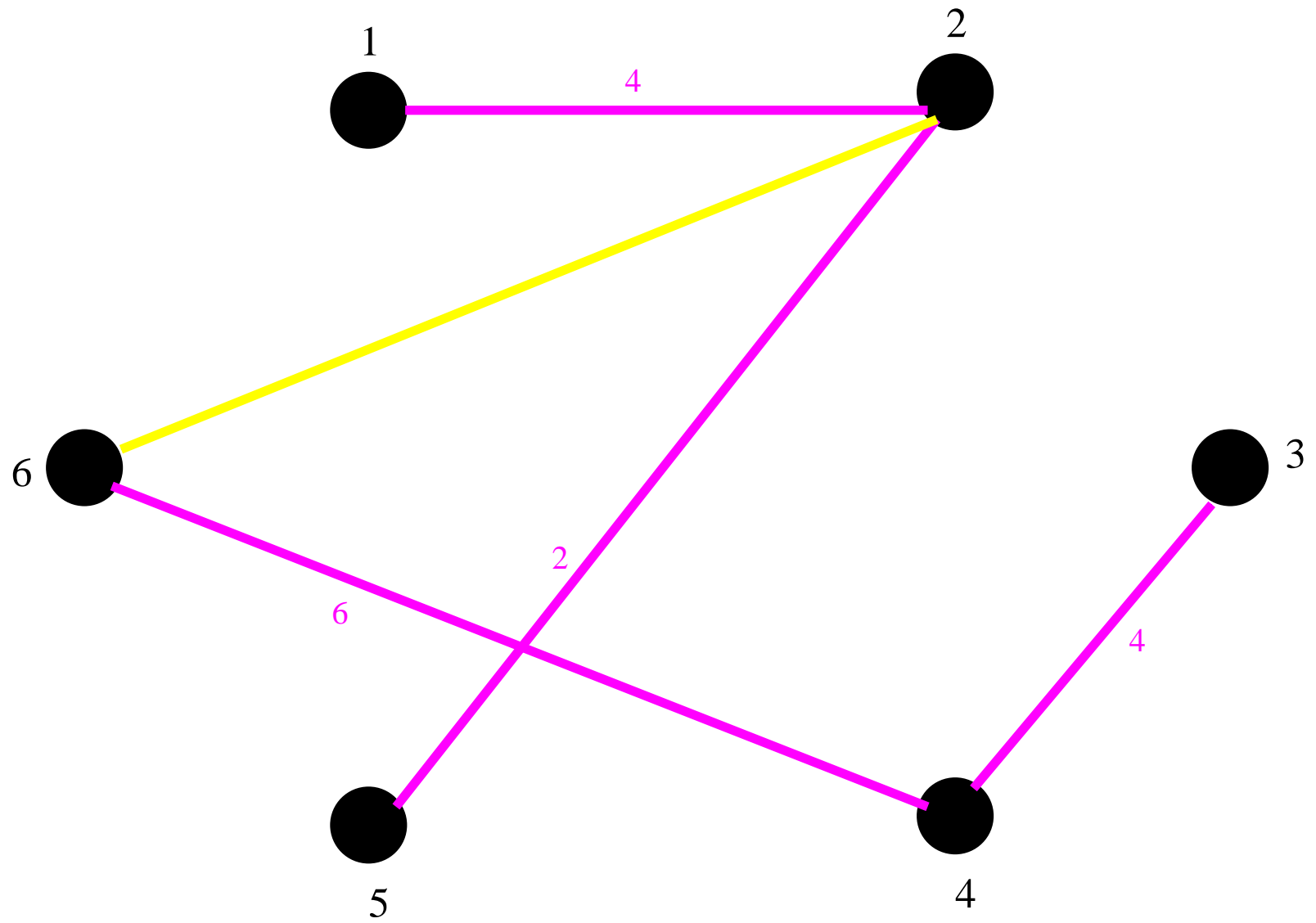


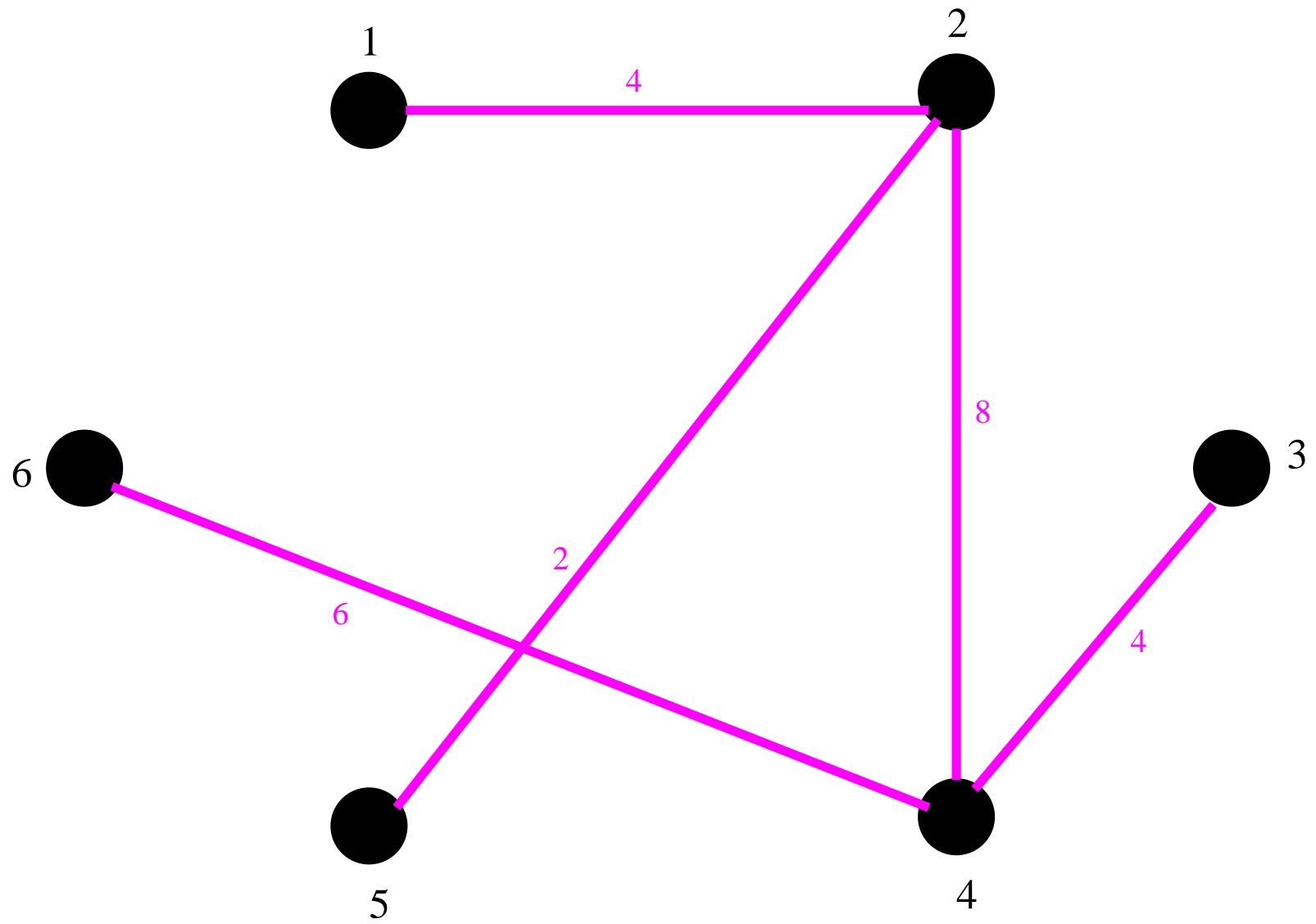


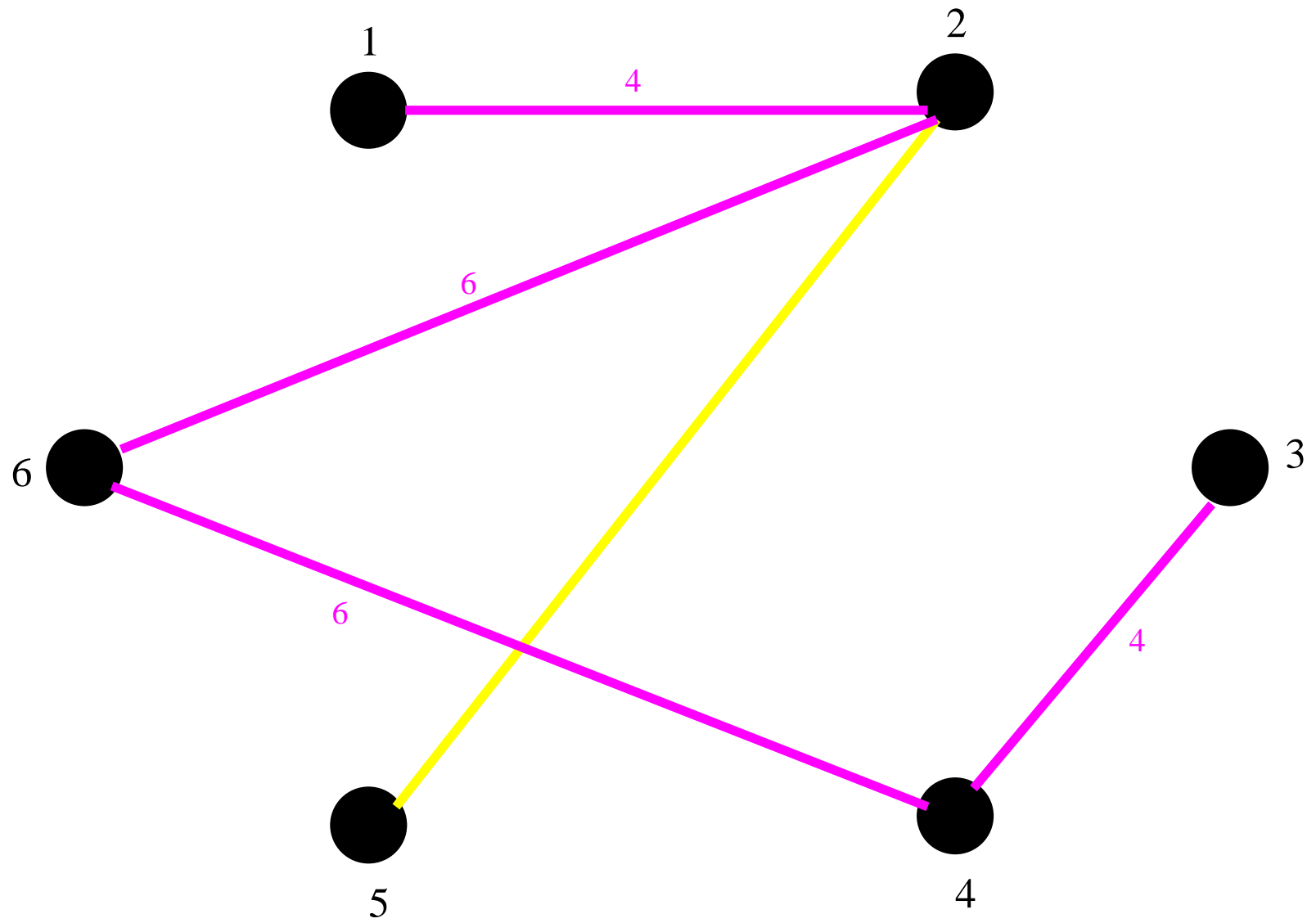


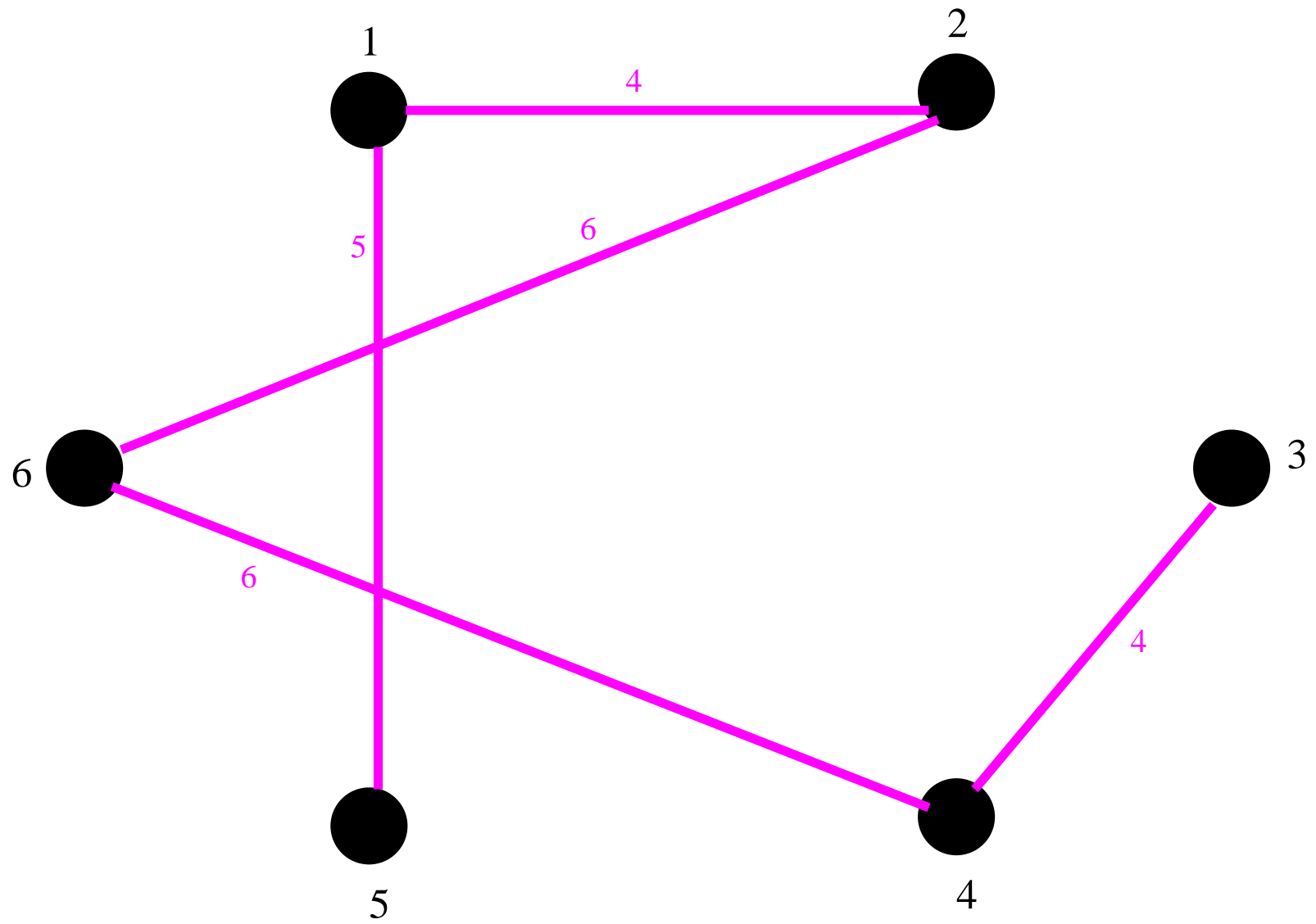


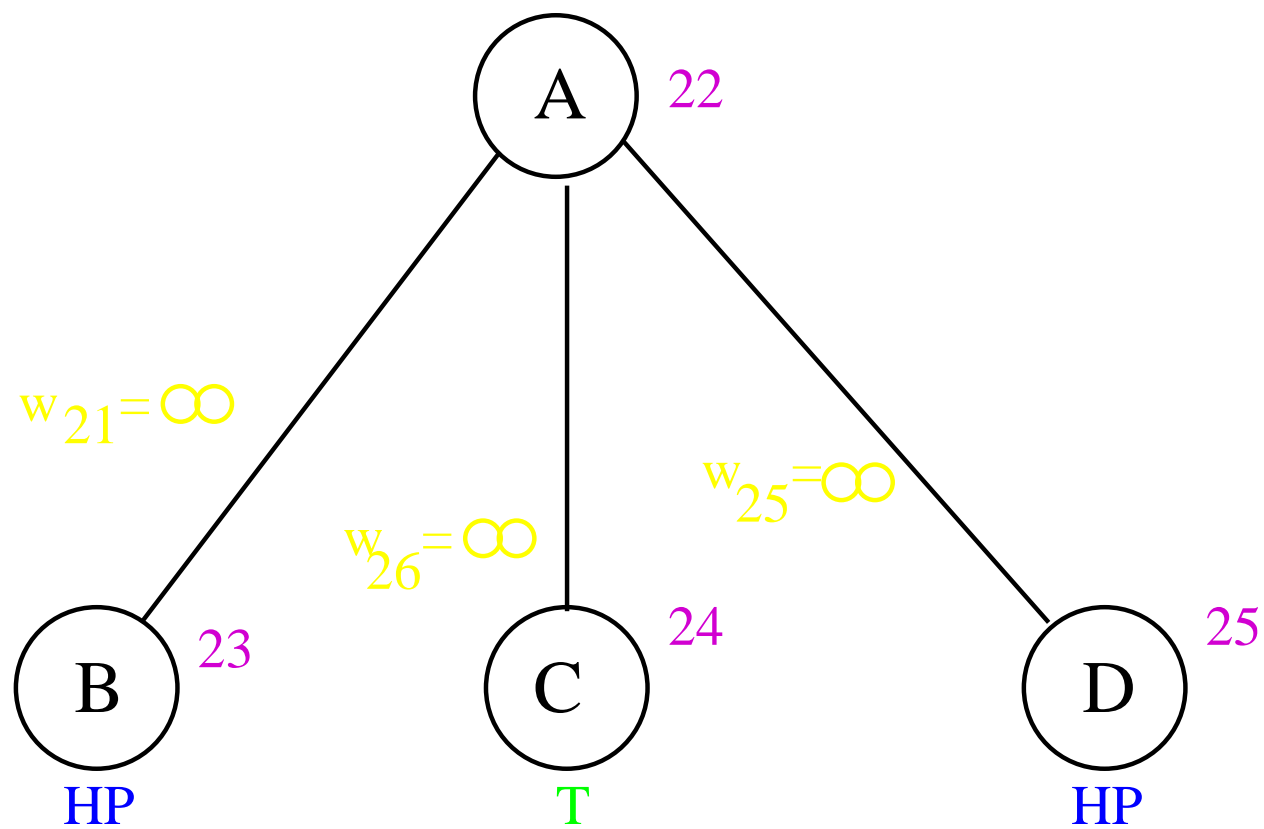












Problem Podróżującego Komiwojażera – rozwiązanie zadania **A**

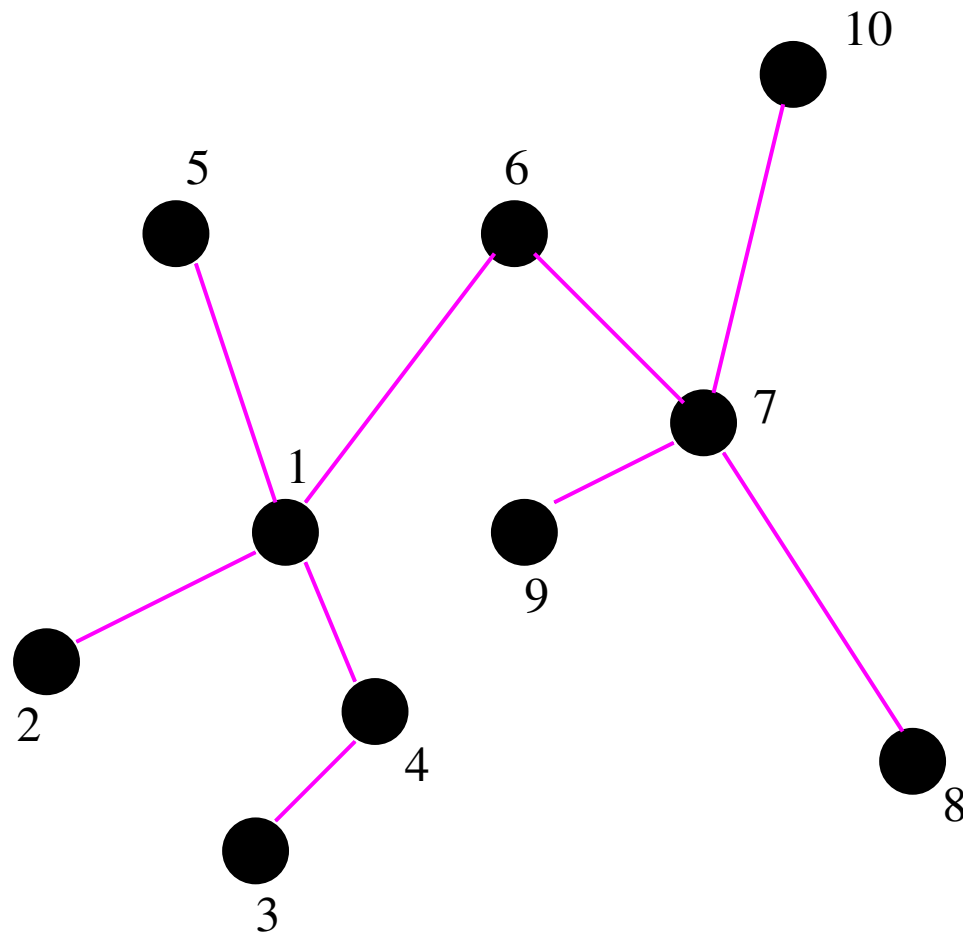
Algorytm nagradzania i karania wierzchołków

- *Znajdujemy minimalne drzewo rozpięte. Jeżeli uzyskane minimalne drzewo jest ścieżką, to STOP.*
- *W przeciwnym razie, korzystając z ostatniego z cytowanych twierdzeń, „nagradzamy” wierzchołki stopnia 1 stosując $p(\cdot)$ ujemne (z wyjątkiem dwóch ustalonych mających być końcami najkrótszej ścieżki H - Zadanie B) „karzemy” wierzchołki stopnia > 2 – stosując $p(\cdot)$ dodatnie.*
- *Powtarzamy procedurę dla otrzymanego grafu G^* .*

Przykład . Wyznaczyć za pomocą algorytmu karania i nagradzania wierzchołków najkrótszą ścieżkę Hamiltona między wierzchołkami 8 i 9 w grafie o macierzy wag:

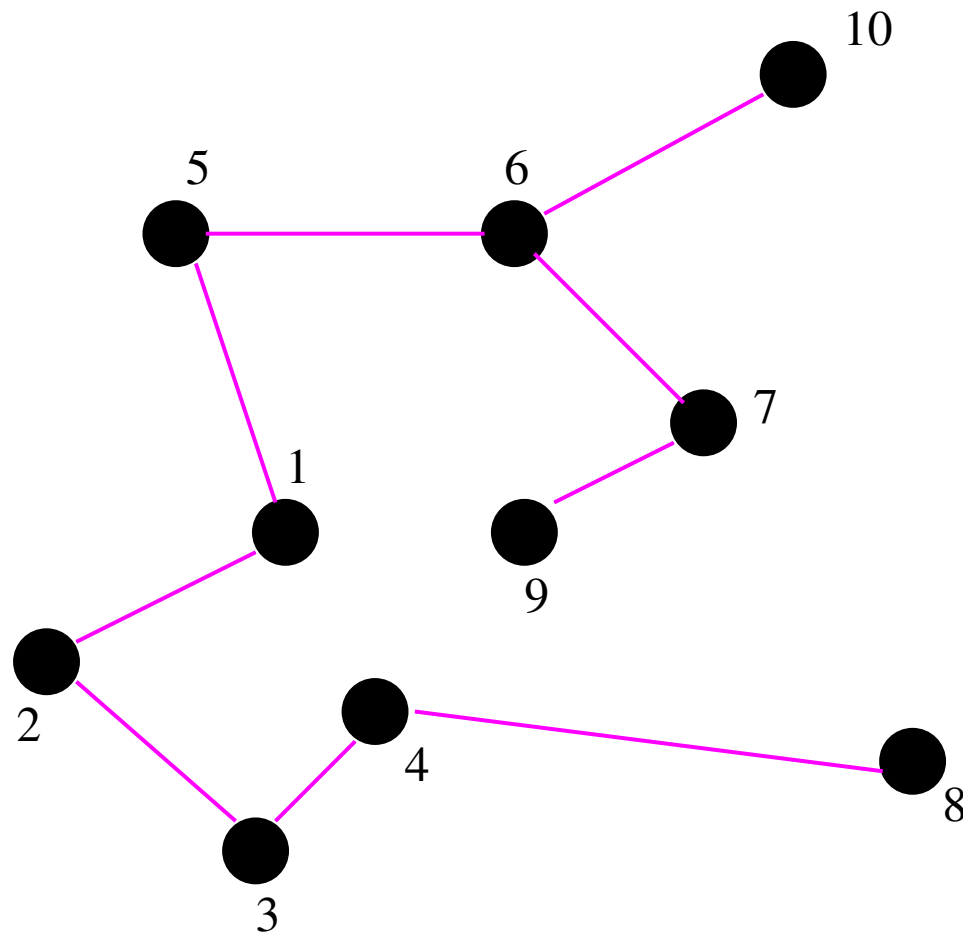
$$W = \begin{pmatrix} 0 & 28 & 31 & 28 & 22 & 36 & 50 & 67 & 40 & 74 \\ 28 & 0 & 31 & 40 & 41 & 64 & 74 & 80 & 63 & 101 \\ 31 & 31 & 0 & 14 & 53 & 53 & 53 & 50 & 42 & 83 \\ 28 & 40 & 14 & 0 & 50 & 41 & 39 & 41 & 28 & 69 \\ 22 & 41 & 53 & 50 & 0 & 40 & 61 & 86 & 53 & 78 \\ 36 & 64 & 53 & 41 & 40 & 0 & 24 & 58 & 22 & 39 \\ 50 & 74 & 53 & 39 & 61 & 24 & 0 & 37 & 11 & 30 \\ 67 & 80 & 50 & 41 & 86 & 58 & 37 & 0 & 36 & 60 \\ 40 & 63 & 42 & 28 & 53 & 22 & 11 & 36 & 0 & 41 \\ 74 & 101 & 83 & 69 & 78 & 39 & 30 & 60 & 41 & 0 \end{pmatrix} .$$

0	28	31	28	22	36	50	1067	1040	74
28	0	31	40	41	64	74	1080	1063	101
31	31	0	14	53	53	53	1050	1042	83
28	40	14	0	50	41	39	1041	1028	69
22	41	53	50	0	40	61	1086	1053	78
36	64	53	41	40	0	24	1058	1022	39
50	74	53	39	61	24	0	10037	1011	30
1067	1080	1050	1041	1086	1058	1037	0	2036	1060
1040	1063	1042	1028	1053	1022	1011	2036	0	1041
74	101	83	69	78	39	30	1060	1041	0



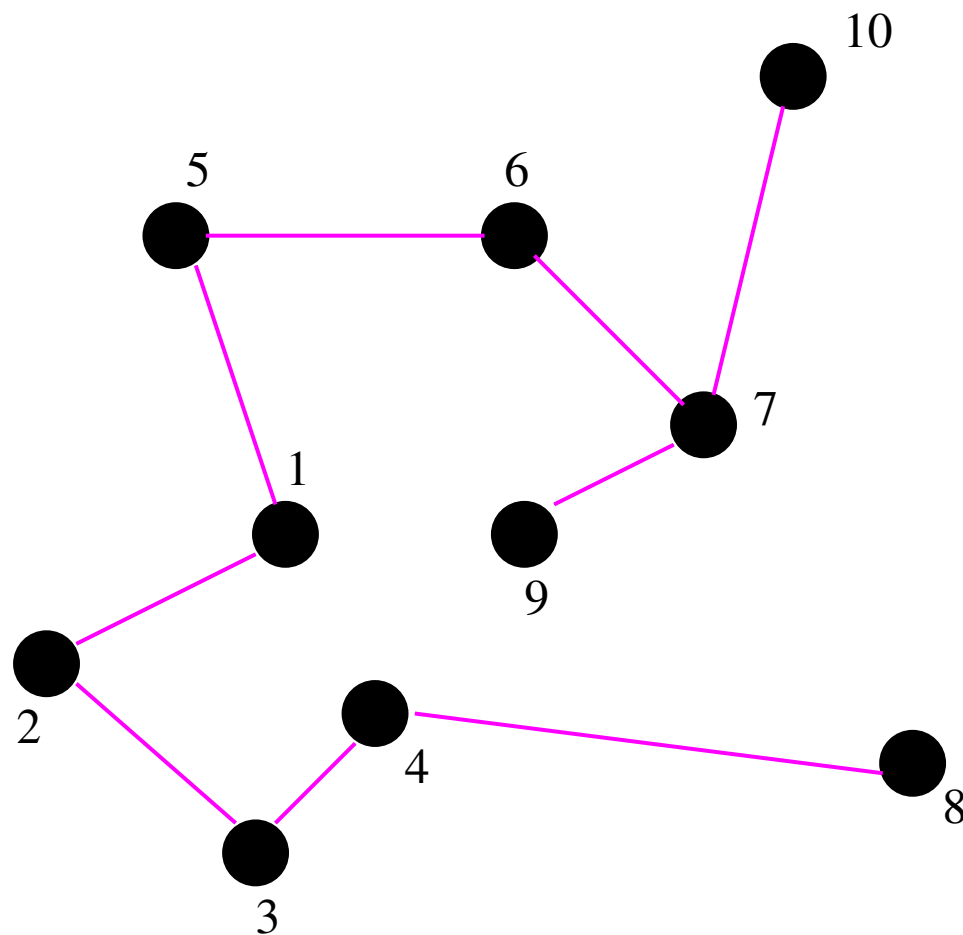
$$p(i) = 5(d_i^T - 2)$$

$$\begin{aligned}
 p(1) &= 10 & p(2) &= -5 & p(3) &= -5 & p(4) &= 0 & p(5) &= -5 \\
 p(6) &= 0 & p(7) &= 10 & p(8) &= -5 & p(9) &= -5 & p(10) &= -5
 \end{aligned}$$



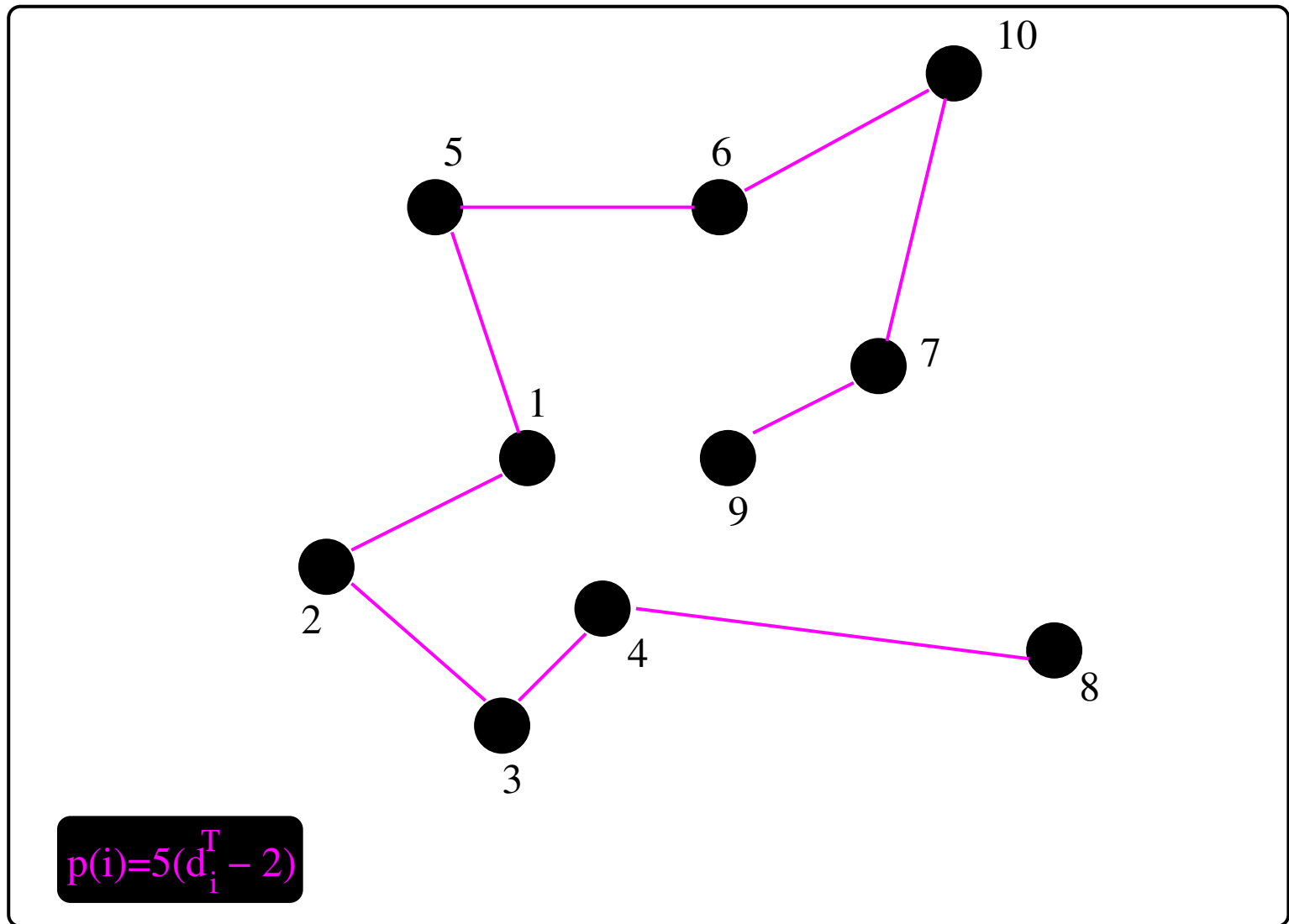
$$p(i) = 5(d_i^T - 2)$$

$$\begin{aligned}
 &p(1)=0 \quad p(2)=0 \quad p(3)=0 \quad p(4)=0 \quad p(5)=0 \\
 &p(6)=5 \quad p(7)=0 \quad p(8)=-5 \quad p(9)=-5 \quad p(10)=-5
 \end{aligned}$$



$$p(i) = 5(d_i^T - 2)$$

$p(1)=0$ $p(2)=0$ $p(3)=0$ $p(4)=0$ $p(5)=0$
 $p(6)=0$ $p(7)=5$ $p(8)=-5$ $p(9)=-5$ $p(10)=-5$



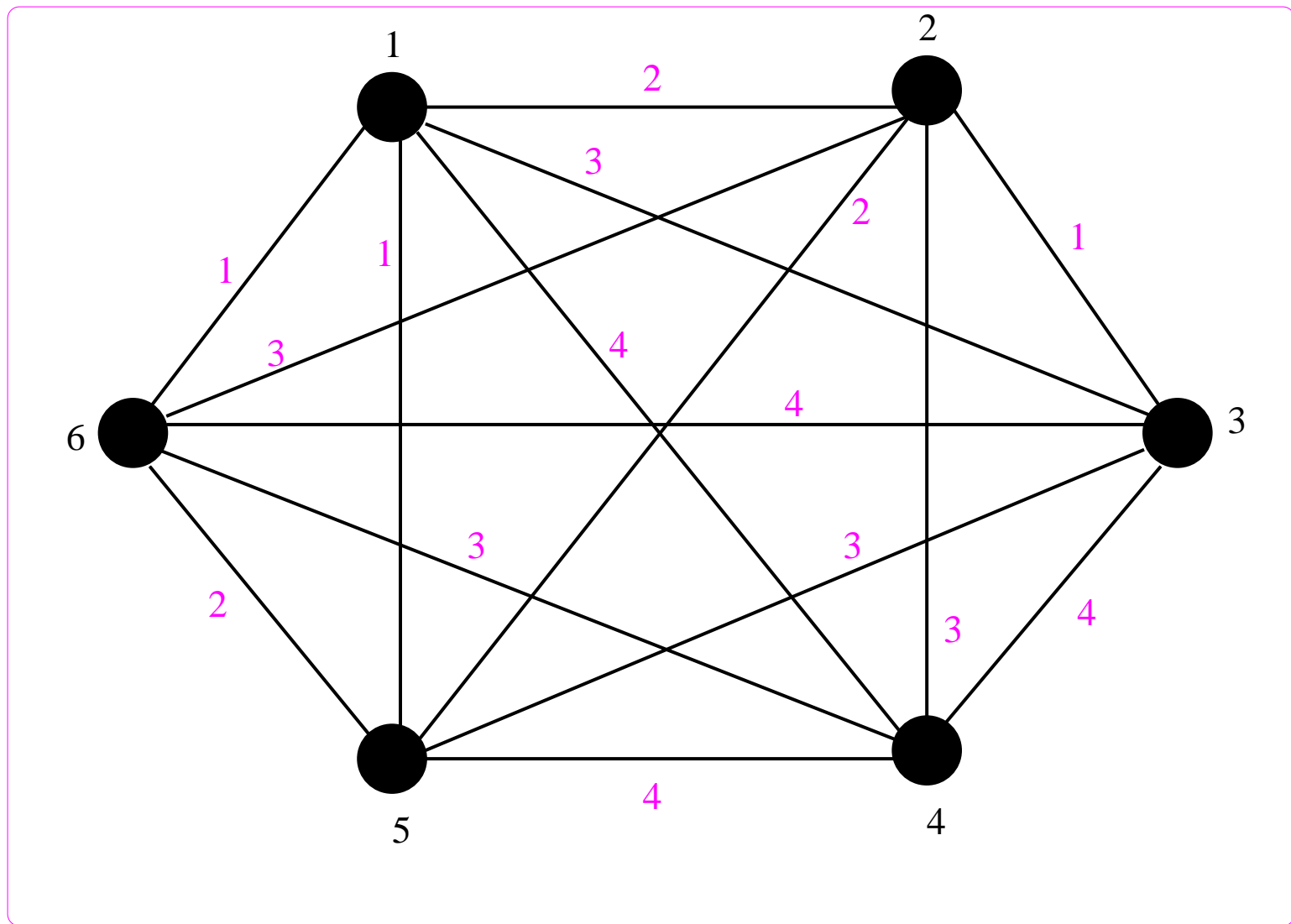
9.3 Algorytmy aproksymacyjne dla TSP

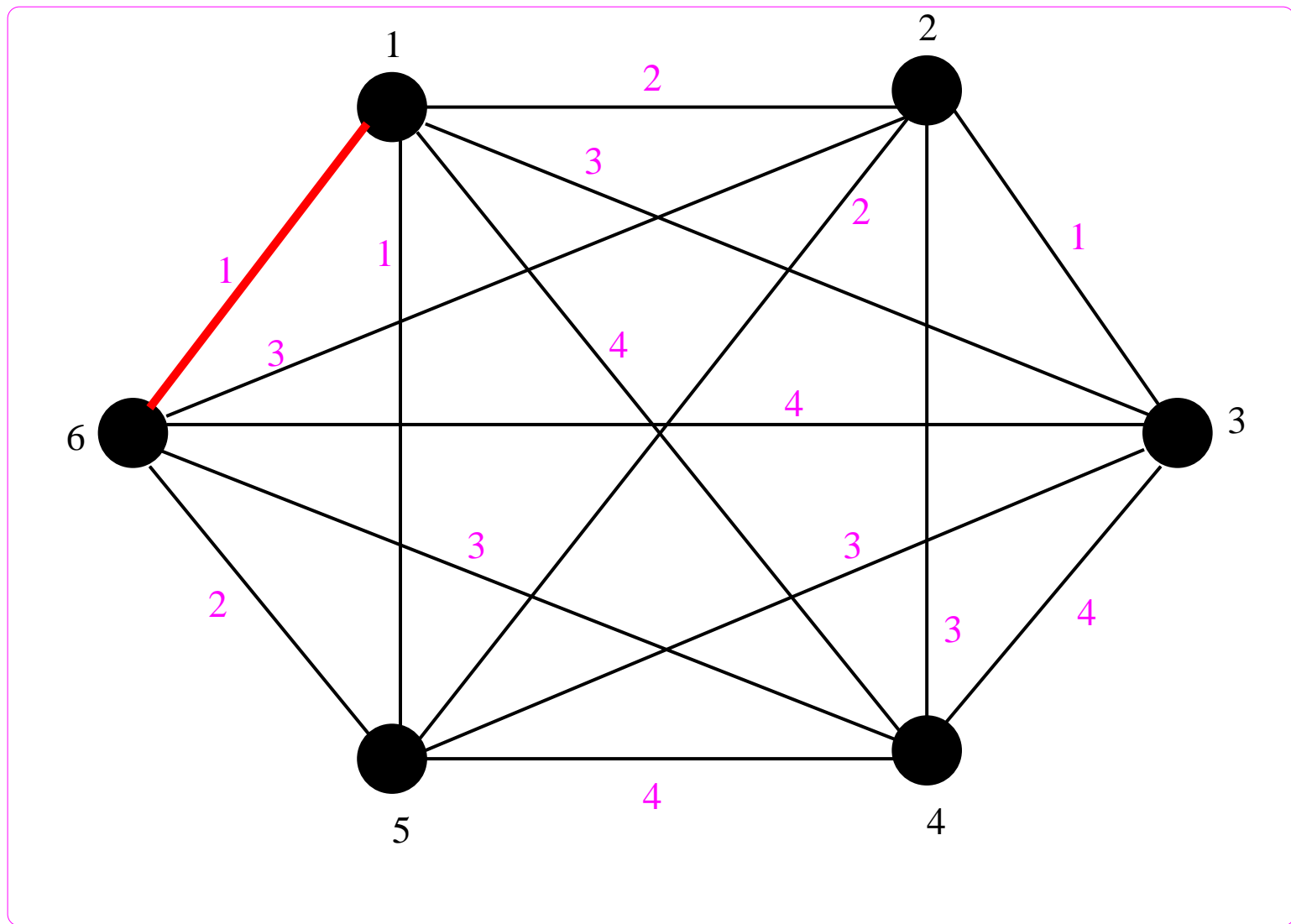
- dla wielu problemów trudnych obliczeniowo, takich jak na przykład problem TSP , zamiast szukać rozwiązania dokładnego skłaniamy się często do poszukiwania rozwiązania przybliżonego
- w tym celu konstruujemy **algorytmy aproksymacyjne**
- jeżeli R jest rozwiązaniem uzyskanym za pomocą takiego algorytmu, natomiast R_0 jest rozwiązaniem dokładnym, to parametrem określającym skuteczność jego działania jest współczynnik aproksymacji określany jako najmniejsza wartość α taka, że $1 \leq R/R_0 \leq \alpha$

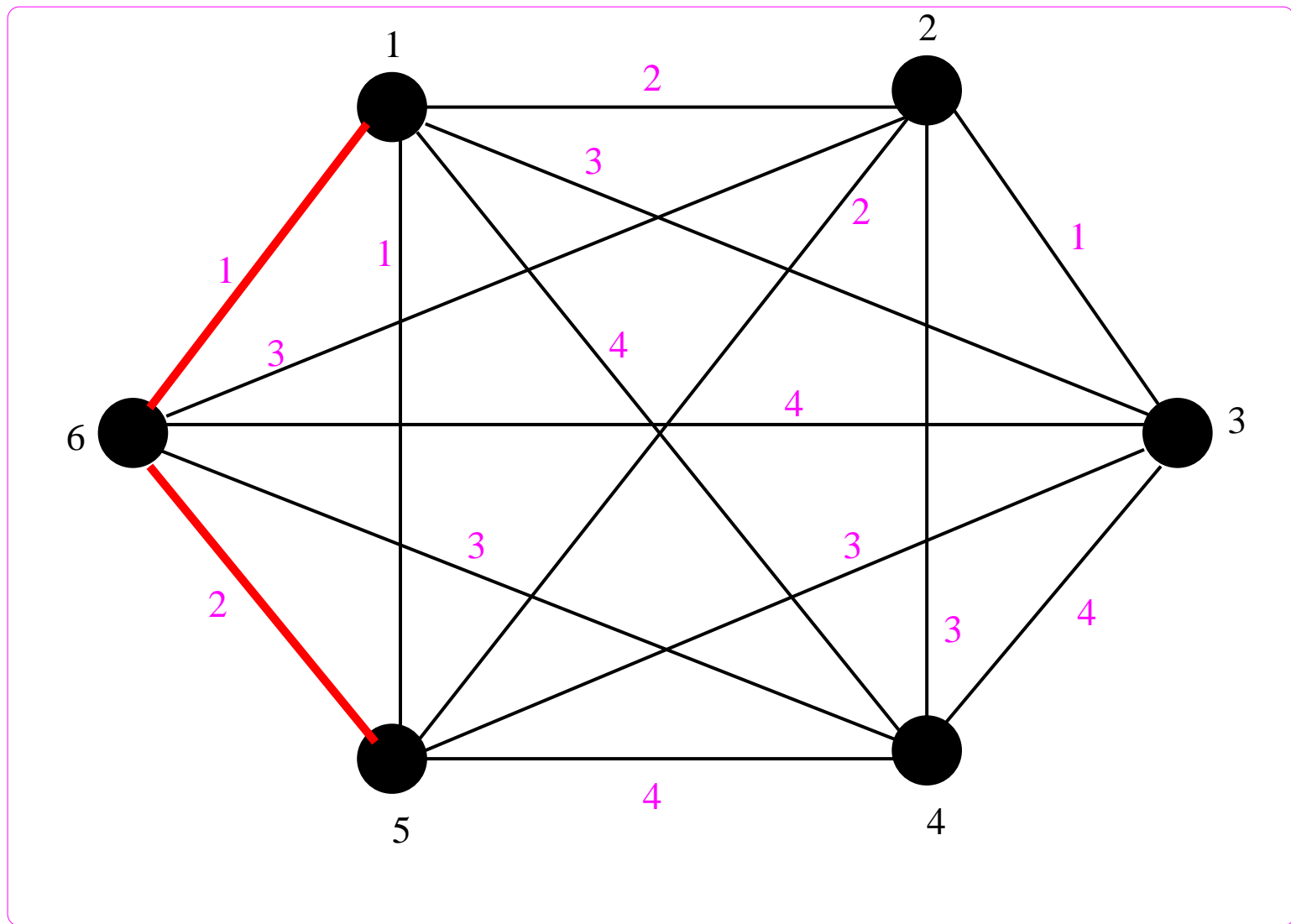
“Naiwny” algorytm dla TSP

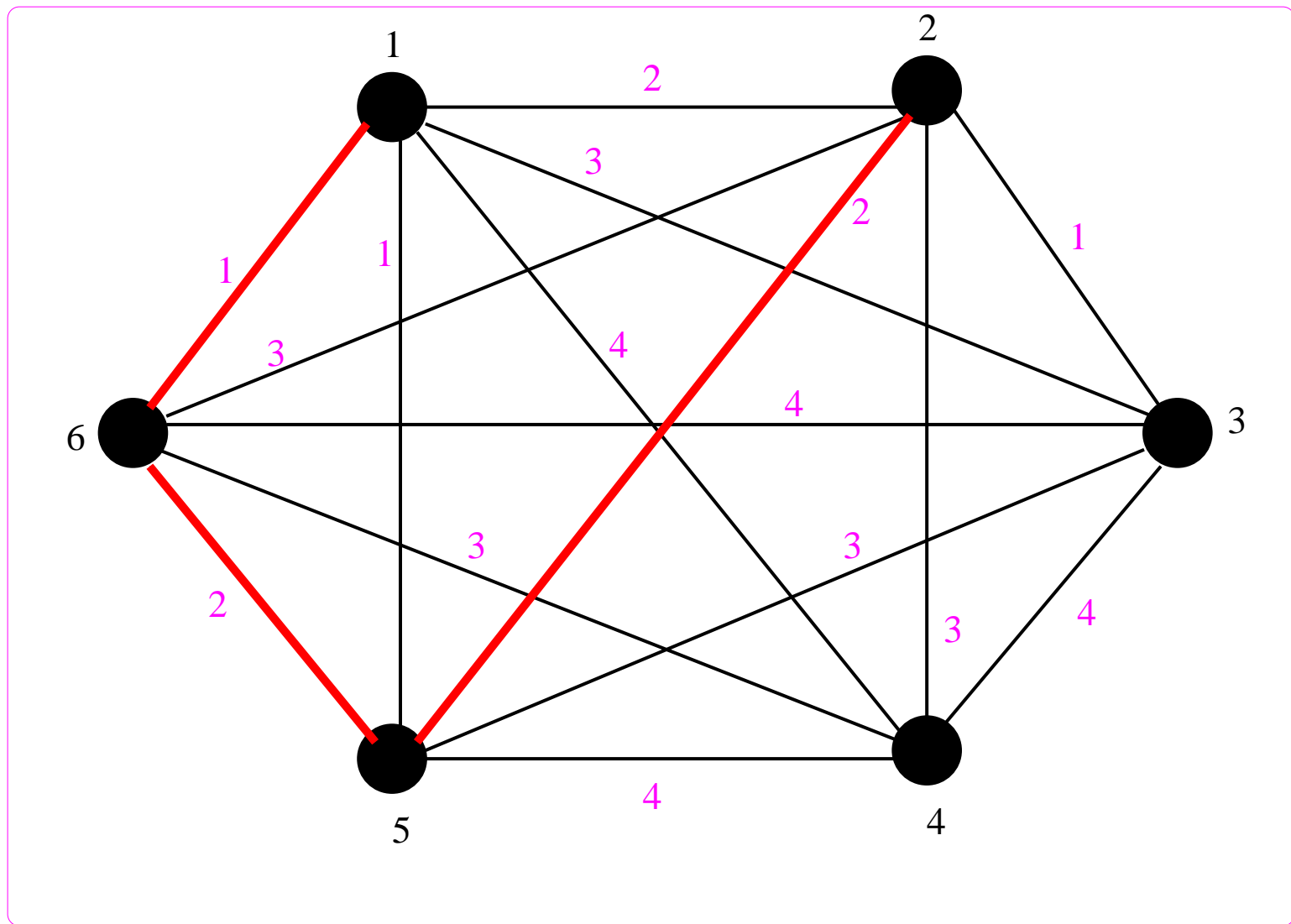
1. Rozpocznij od dowolnego wierzchołka i oznacz go v_1 , znajdź najkrótszą krawędź wychodzącą z v_1 i oznacz koniec tej krawędzi przez v_2 .
2. Wyjdź z wierzchołka v_2 i przejdź do najbliższego wierzchołka którego dodanie nie zamyka cyklu. Oznacz ten wierzchołek jako v_3 .
3. Kontynuuj tą procedurę aż do chwili gdy utworzysz ścieżkę Hamiltona v_1, v_2, \dots, v_n . Utwórz cykl Hamiltona poprzez dodanie krawędzi (v_n, v_1) .

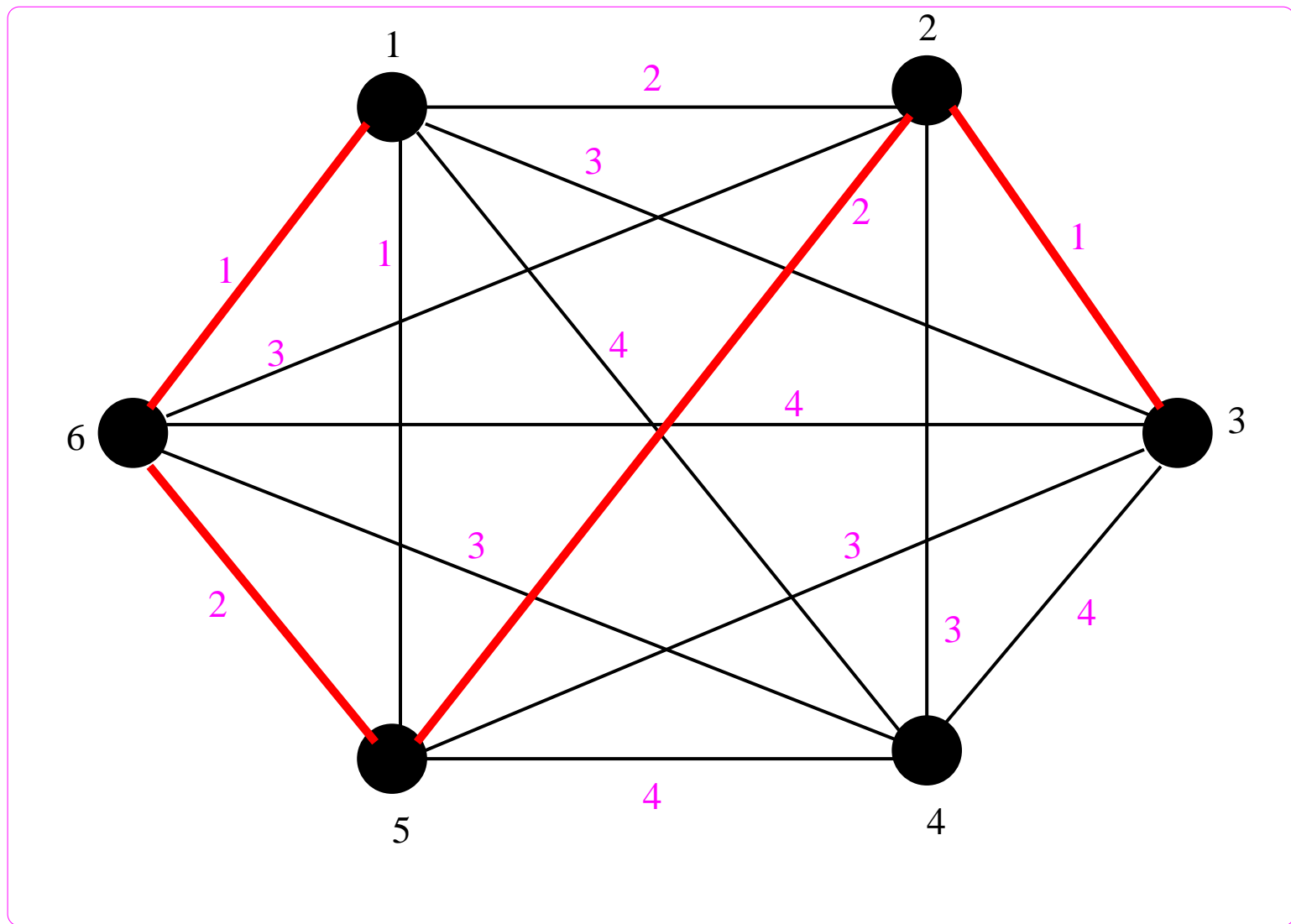
Niestety dla tego algorytmu $\alpha = \frac{1}{2}(\lceil \ln n \rceil + 1)!!$

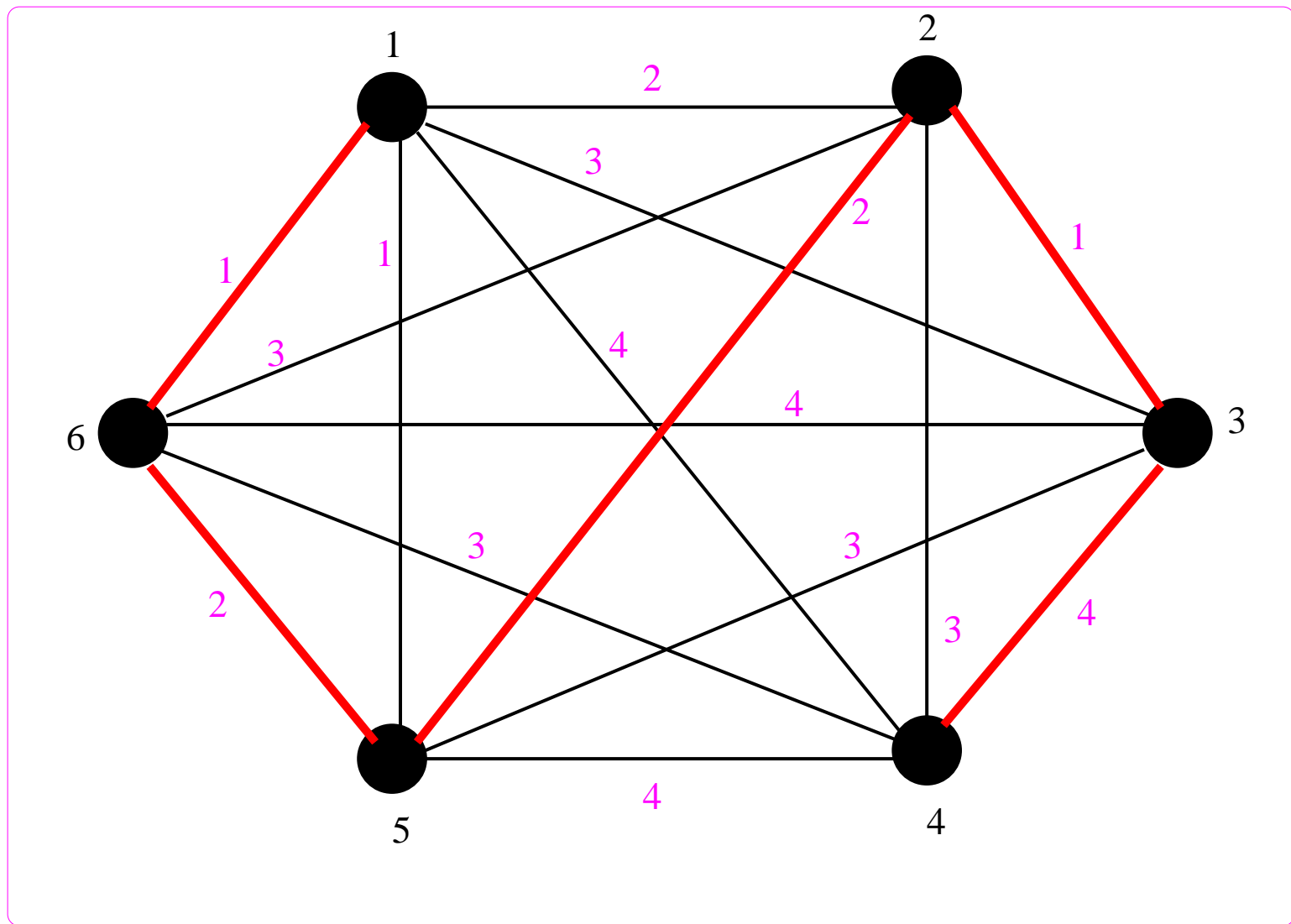


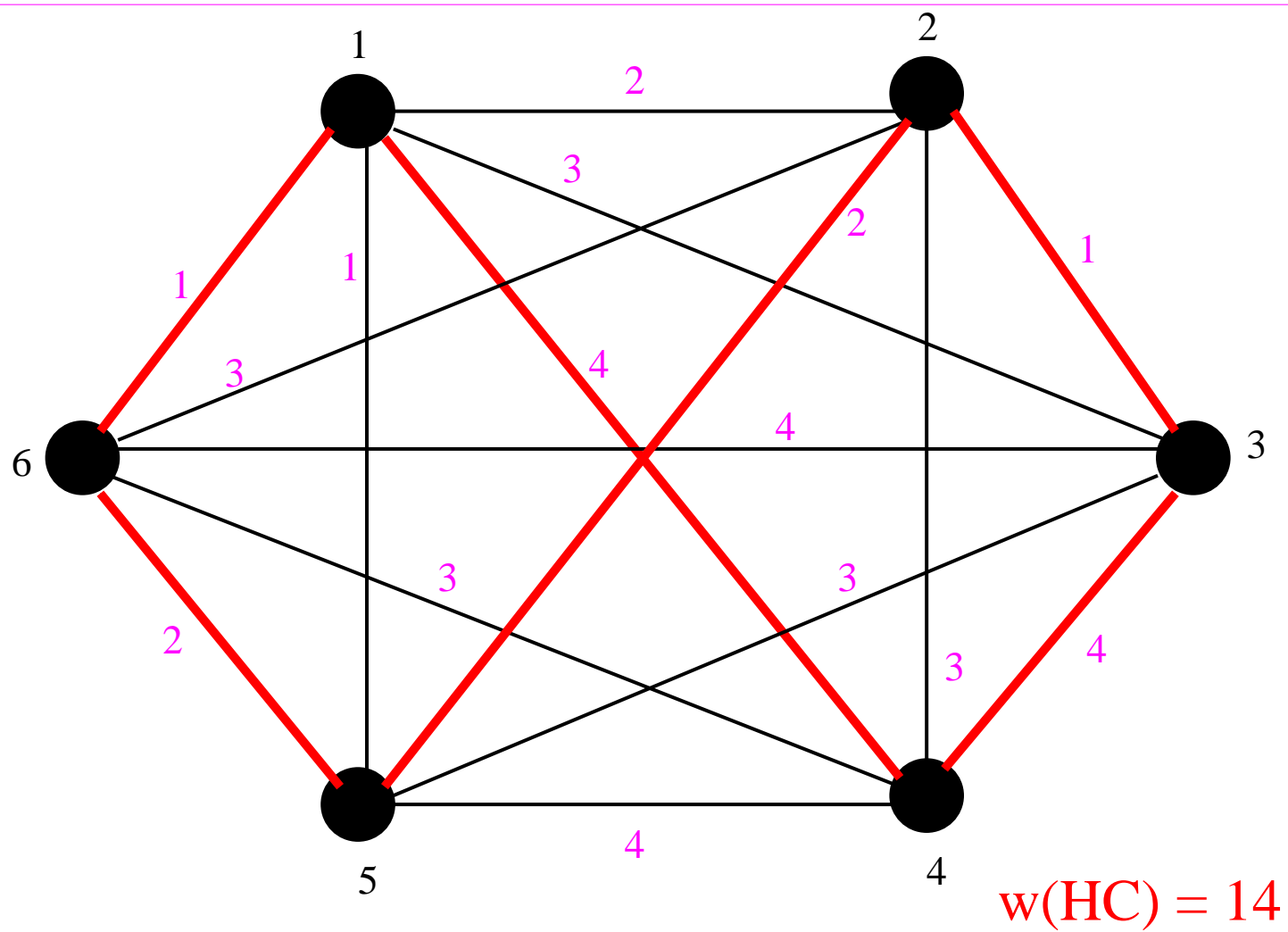












Algorytm “dwa-razy-dookoła-minimalnego-rozpiętego-drzewa” dla TSP

1. Znajdź minimalne rozpięte drzewo T w grafie G .
2. Przeszukaj drzewo T metodą DFS (przeszukiwania w głąb) i przypisz każdemu wierzchołkowi v drzewa T etykietę $L(v)$ równą czasowi odwiedzin wierzchołka v w tym przeszukaniu
3. Podaj na wyjściu przybliżenie cyklu Hamiltona o najmniejszej długości następującej postaci:

$$C = (v_{i_1}, v_{i_2}, \dots, v_{i_n}, v_{i_1}), \text{ gdzie } L(v_{i_j}) = j$$

Twierdzenie .

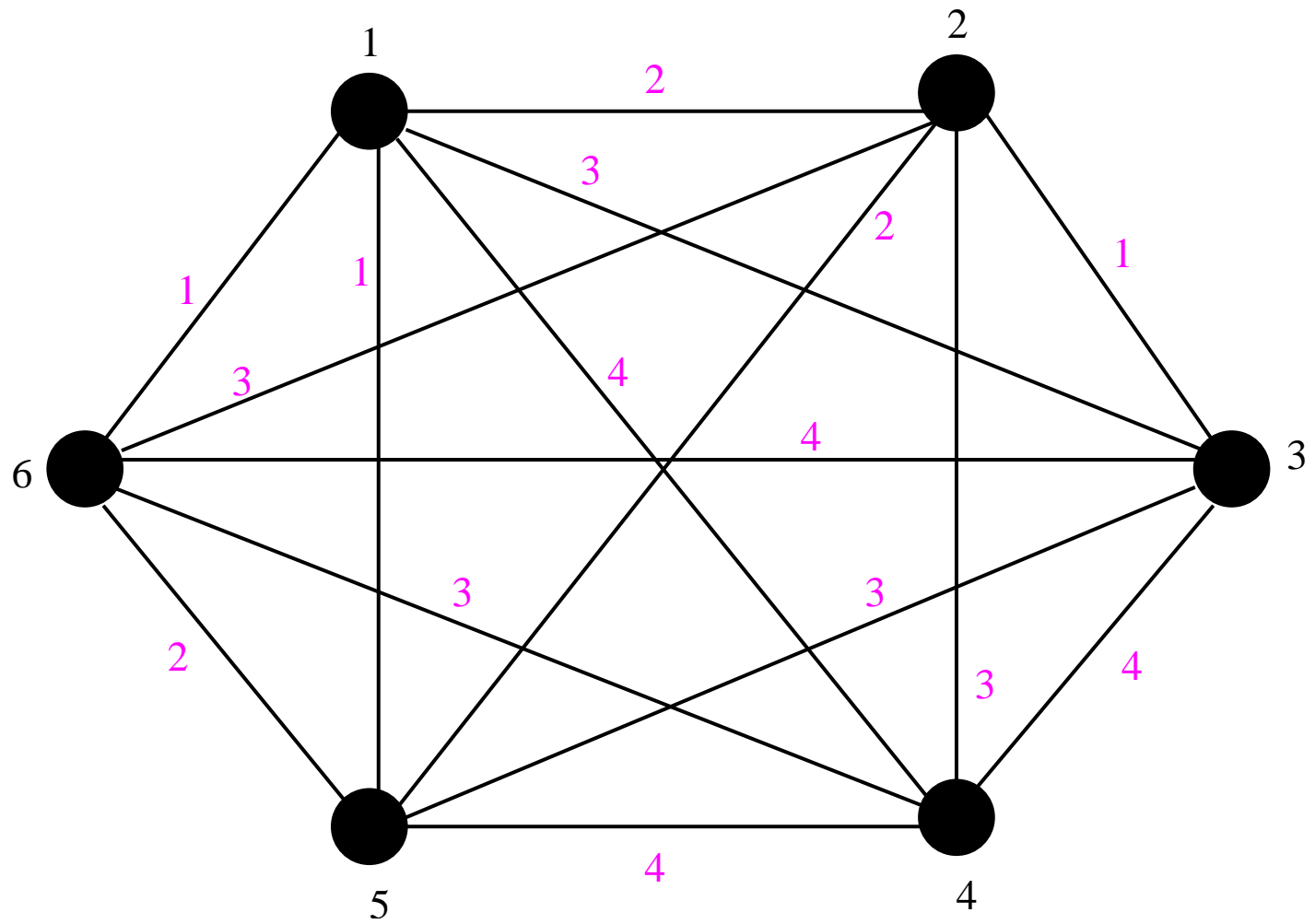
Dla dowolnego grafu w którym wagi spełniają nierówność trójkąta, algorytm “dwa-razy-dookoła-minimalnego-rozpiętego-drzewa” znajduje rozwiązanie problemu TSP ze współczynnikiem aproksymacji $\alpha = 2$.

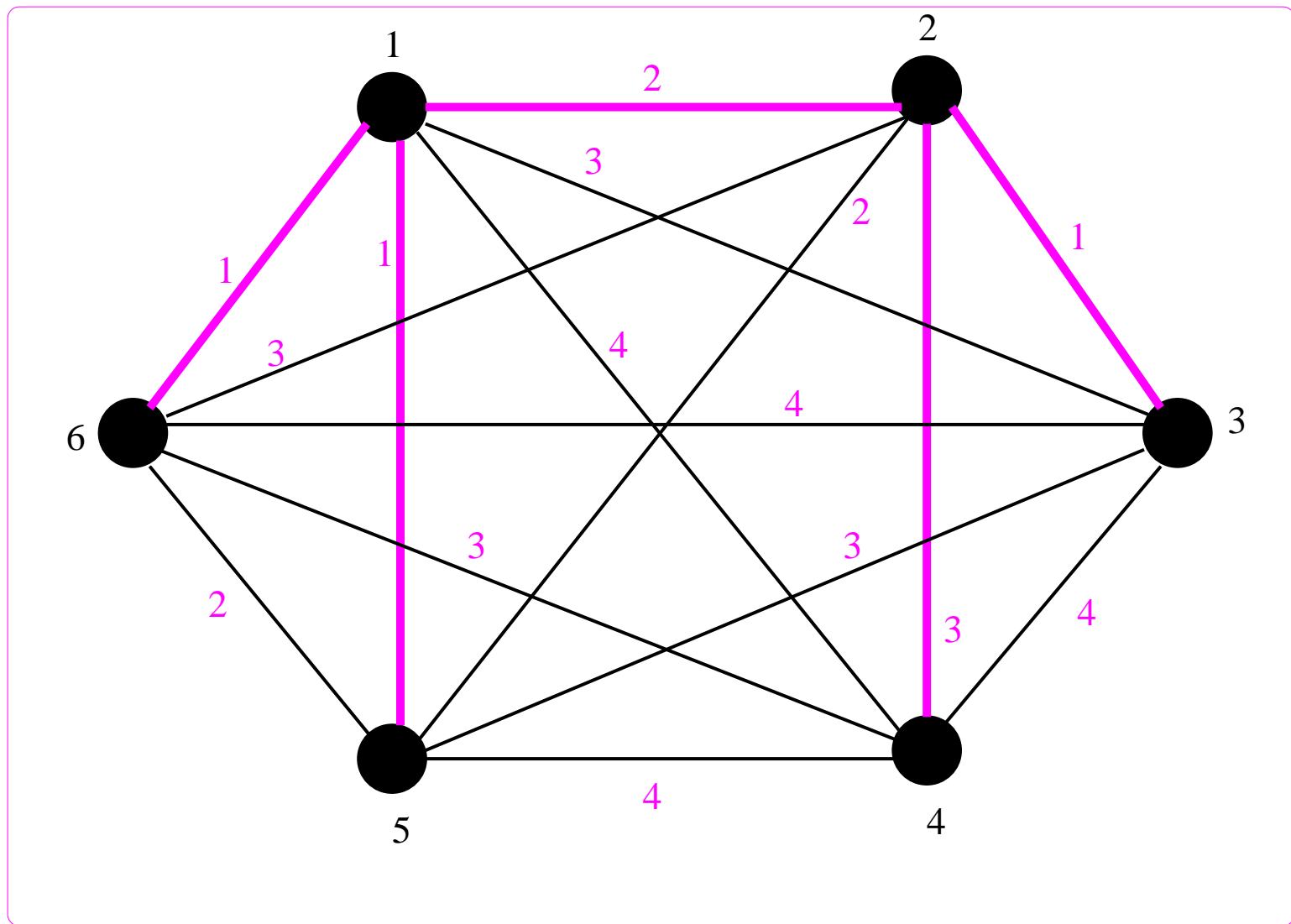
Dowód.

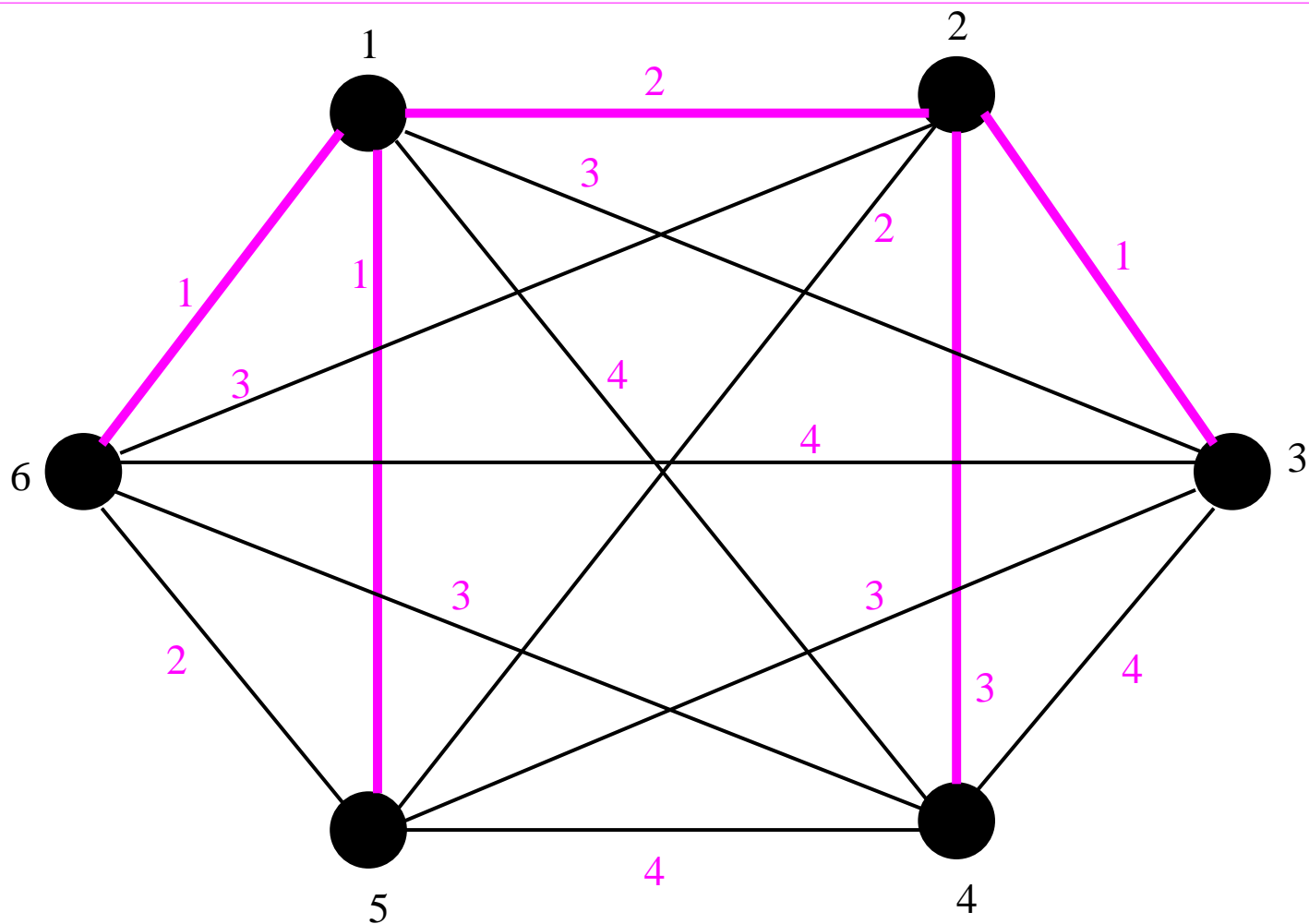
- Niech W będzie wagą minimalnego rozpiętego drzewa w G , a W_0 będzie najkrótszym cyklem Hamiltona w G . Zauważmy, że $W < W_0$, ponieważ rozpięte drzewo krótsze niż najkrótszy cykl Hamiltona otrzymujemy przez usunięcie dowolnej krawędzi cyklu.
- Następnie zauważmy, że DFS rozpiętego drzewa tworzy domknięty spacer C_0 przechodzący dwukrotnie przez każdą krawędź tego drzewa. Jeżeli drzewo T jest minimalnym rozpiętym drzewem, to C_0 ma długość równą $2W$, która jest ostro mniejsza od $2W_0$.

Dowód. c.d.

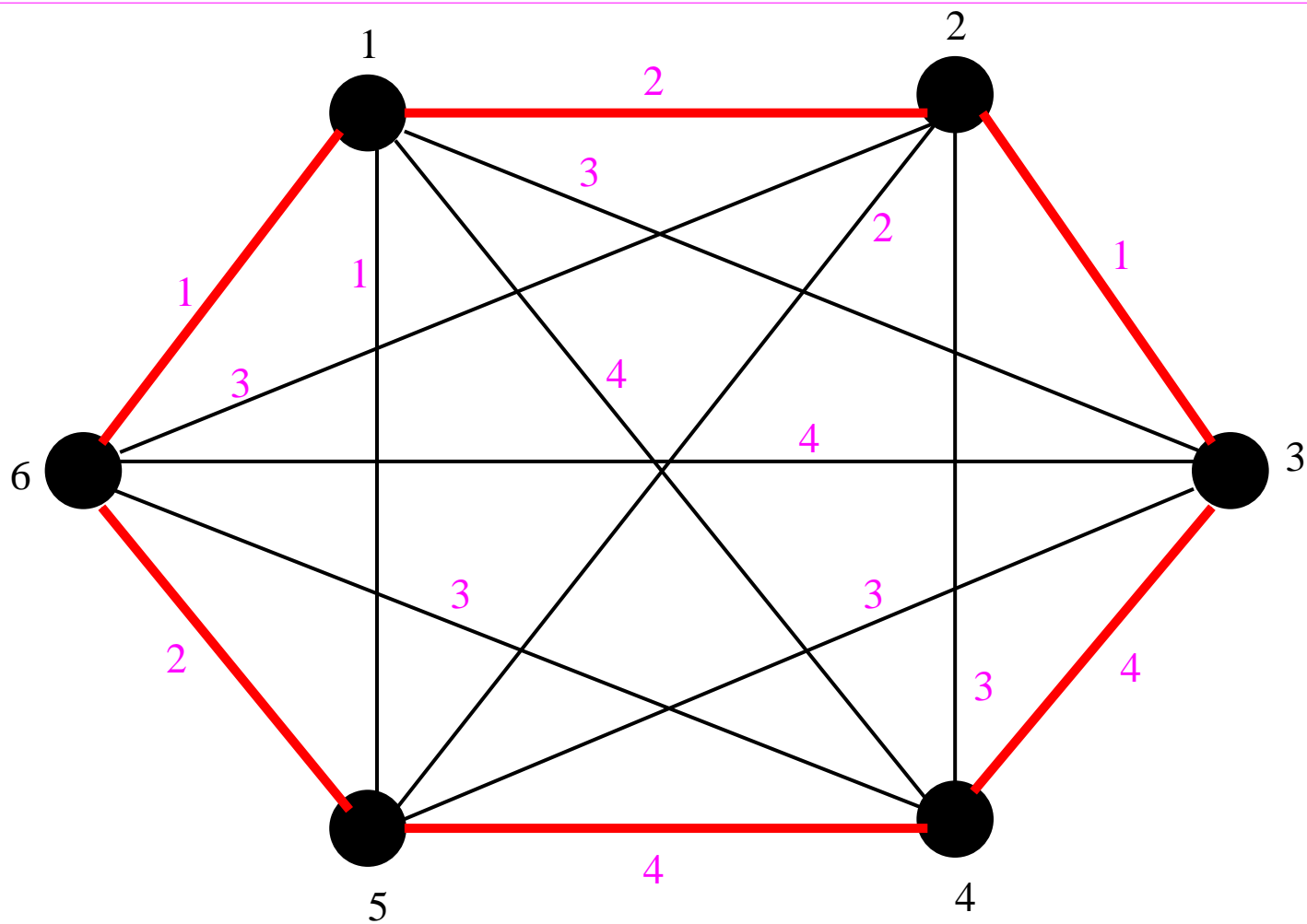
- Cykl C generowany przez domknięty spacer C_0 , przechodzi przez wierzchołki w takiej samej kolejności w jakiej C_0 je odwiedza, z tą różnicą, że C przechodzi do kolejnego, jeszcze nieodwiedzonego wierzchołka, bezpośrednio a nie jak C_0 , który “rewizytuje” każdy wierzchołek. Ponieważ wagi krawędzi grafu G spełniają nierówność trójkąta, cykl C nie może być dłuższy od C_0 , co kończy dowód twierdzenia.







$C_0 = (1, 2, 3, 2, 4, 2, 1, 5, 1, 6, 1)$



$C = (1, 2, 3, 4, 5, 6, 1)$

$w(HC) = 14$

Algorytm “skojarzenia o minimalnej wadze” dla TSP

1. Znajdź minimalne rozpięte drzewo T w grafie G .
2. Ustal zbiór V' wierzchołków stopnia nieparzystego w T i znajdź skojarzenie o minimalnej wadze M dla V' .
3. Skonstruuj graf eulerowski G' poprzez dodanie krawędzi z M do drzewa T .
4. Znajdź obchód Eulera C_0 grafu G' i nadaj etykiety $L(v)$ poszczególnym wierzchołkom, w takiej kolejności w jakiej wierzchołki były po raz pierwszy odwiedzone w tym obchodzie.
5. Podaj na wyjściu przybliżenie cyklu Hamiltona o najmniejszej długości następującej postaci:

$$C = (v_{i_1}, v_{i_2}, \dots, v_{i_n}, v_{i_1}), \text{ gdzie } L(v_{i_j}) = j$$

Twierdzenie .

Dla dowolnego grafu w którym wagi spełniają nierówność trójkąta, algorytm “skojarzenia o minimalnej wadze” znajduje rozwiązanie problemu TSP ze współczynnikiem aproksymacji $\alpha = 3/2$.

Dowód.

- Zauważmy, że z faktu iż w grafie G wagi krawędzi spełniają nierówność trójkąta cykl C nie może być dłuższy od obchodu C_0 . Ponadto, tak jak w poprzednim algorytmie, C przechodzi przez wierzchołki w takiej samej kolejności w jakiej C_0 je odwiedza, z tą różnicą, że C przechodzi do kolejnego, jeszcze nieodwiedzonego wierzchołka, bezpośrednio a nie jak C_0 , który “rewizytuje” każdy wierzchołek.

Dowód. c.d.

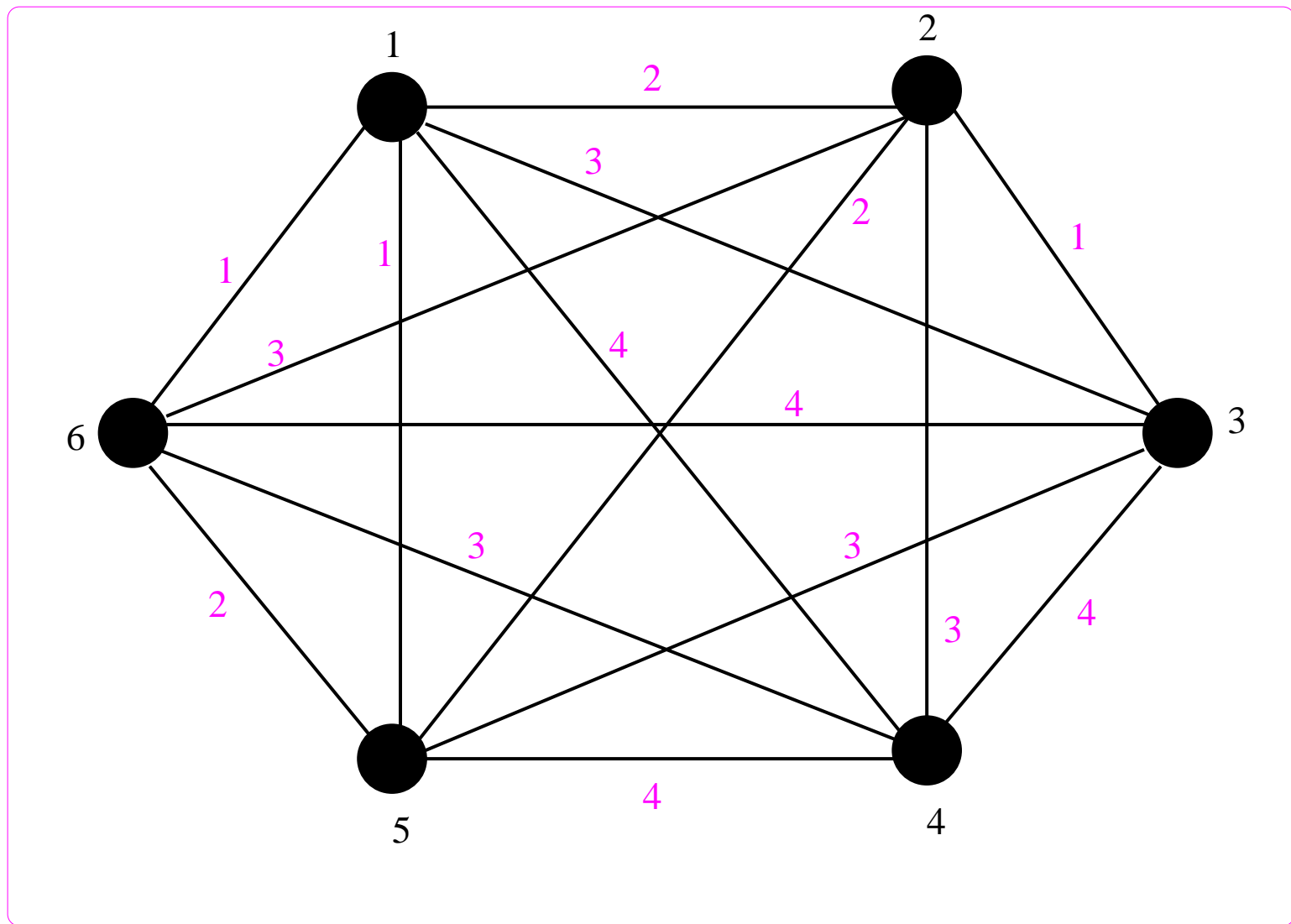
- Waga obchodu C_0 jest równa sumie wag W i W_1 , gdzie W, W_1 to, odpowiednio wagi drzewa T i skojarzenia M .
- Zauważmy, że jeżeli W_0 oznacza wagę najkrótszego cyklu Hamiltona w G , to oczywiście, tak jak poprzednio $W < W_0$.
- Zatem do wykazania, że w naszym przypadku, $\alpha = 3/2$ wystarczy udowodnić, że $W_1 \leq \frac{1}{2}W_0$. W tym celu przypuśćmy, że H jest cyklem Hamiltona o wadze W_0 . Wtedy możemy pokazać istnienie cyklu o długości nie przekraczającej W_0 , który przechodzi jedynie przez wierzchołki z V' . Taki cykl otrzymujemy poprzez trawersowanie cyklu H i “opuszczanie” tych wierzchołków które nie należą do V' . Ponieważ zachodzi nierówność trójkąta, dlatego długość nowego cyklu nie może być większa niż długość cyklu H .

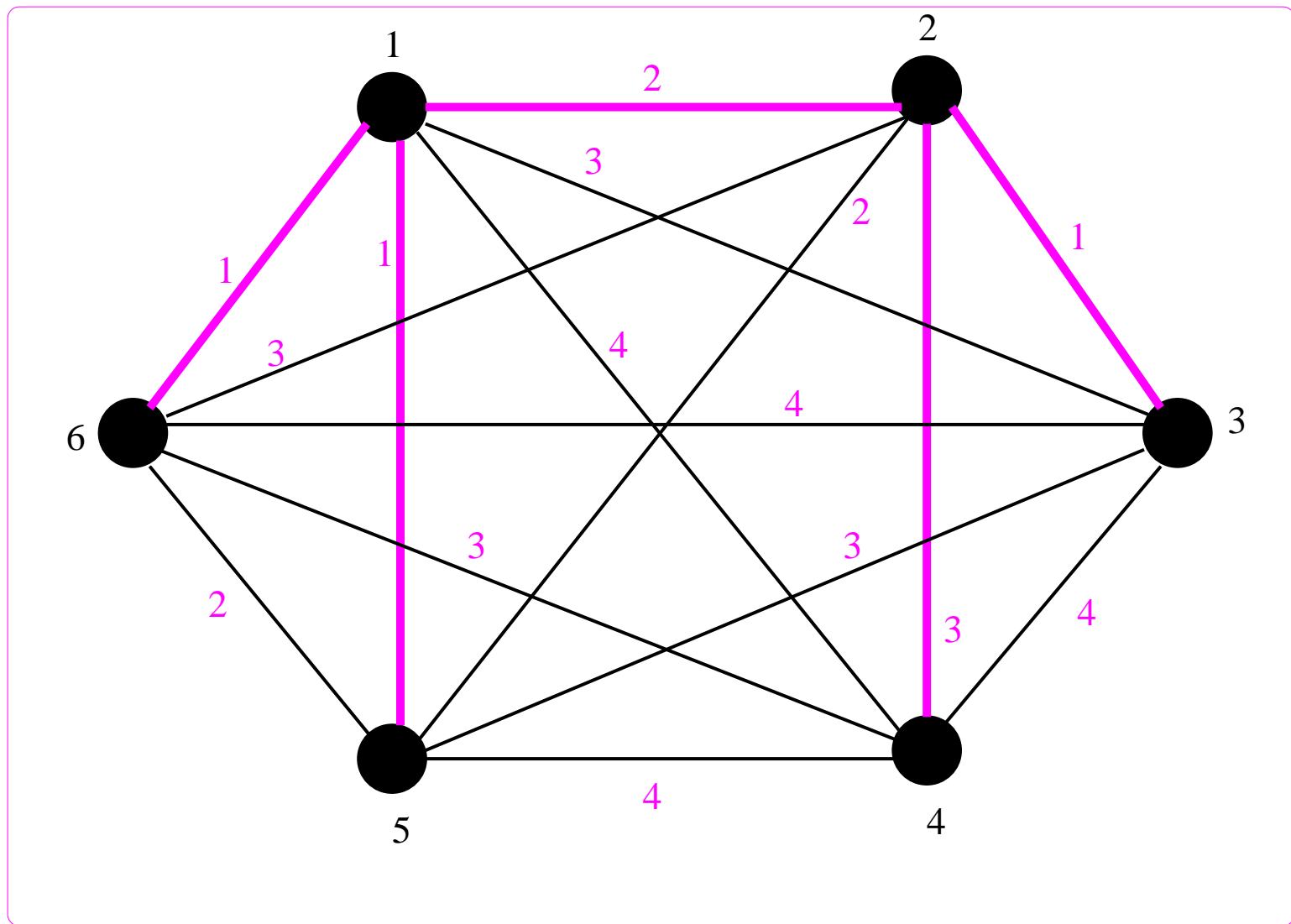
Dowód. c.d.

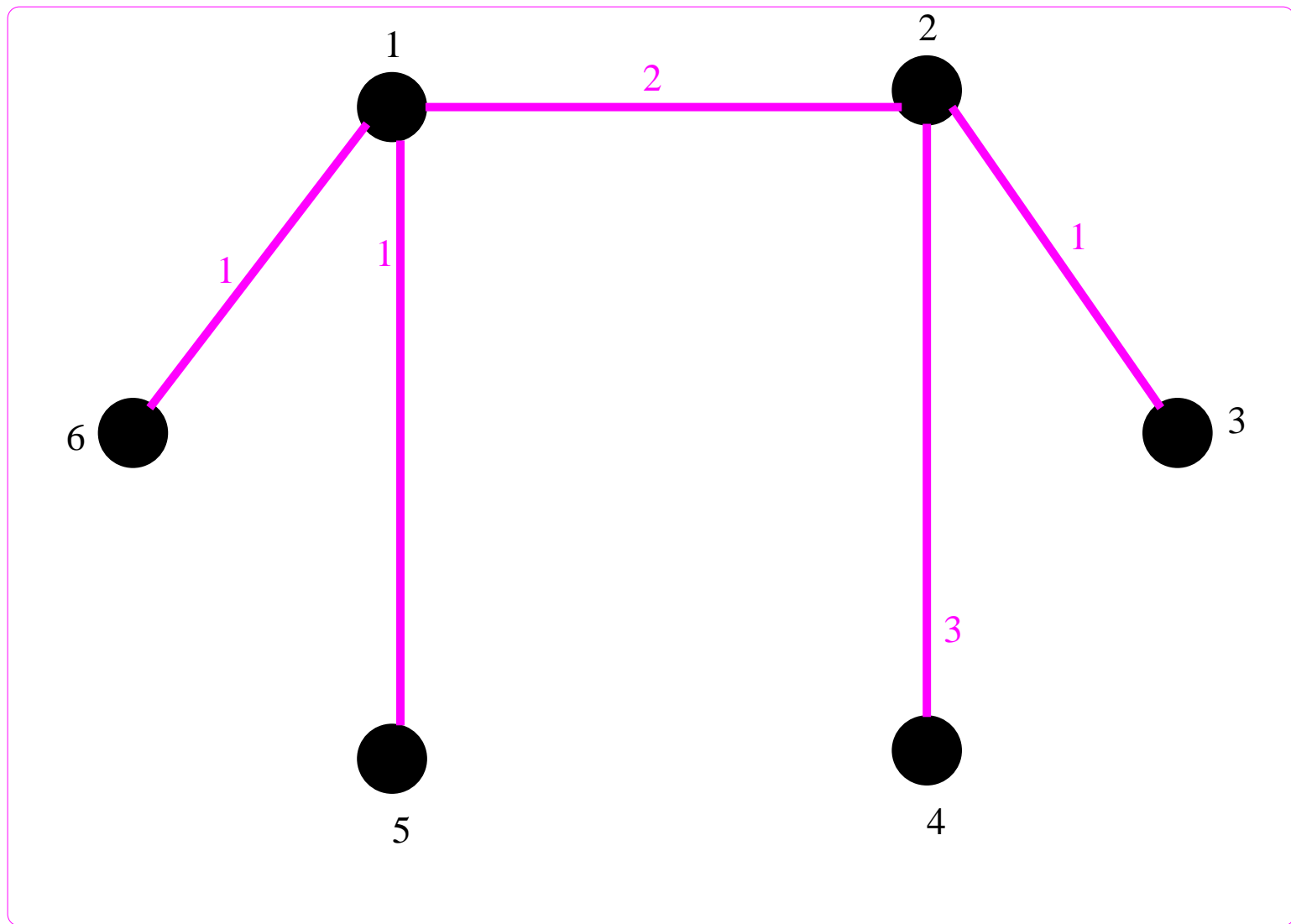
- Zauważmy teraz, że utworzony w powyższy sposób cykl na wierzchołkach V' jest cyklem parzystym. Dlatego można utworzyć w nim dwa skojarzenia, powiedzmy, czerwone i niebieskie, biorąc naprzemiennie krawędzie z tego cyklu. Wybierzmy to skojarzenie które ma mniejsza wagę. Waga tego skojarzenia jest z jednej strony nie mniejsza niż waga W_1 minimalnego skojarzenia M na V' , a z drugiej strony nie przekracza połowy wagi cyklu przechodzącego przez wierzchołki V' . Stąd $W_1 \leq \frac{1}{2}W_0$, czyli

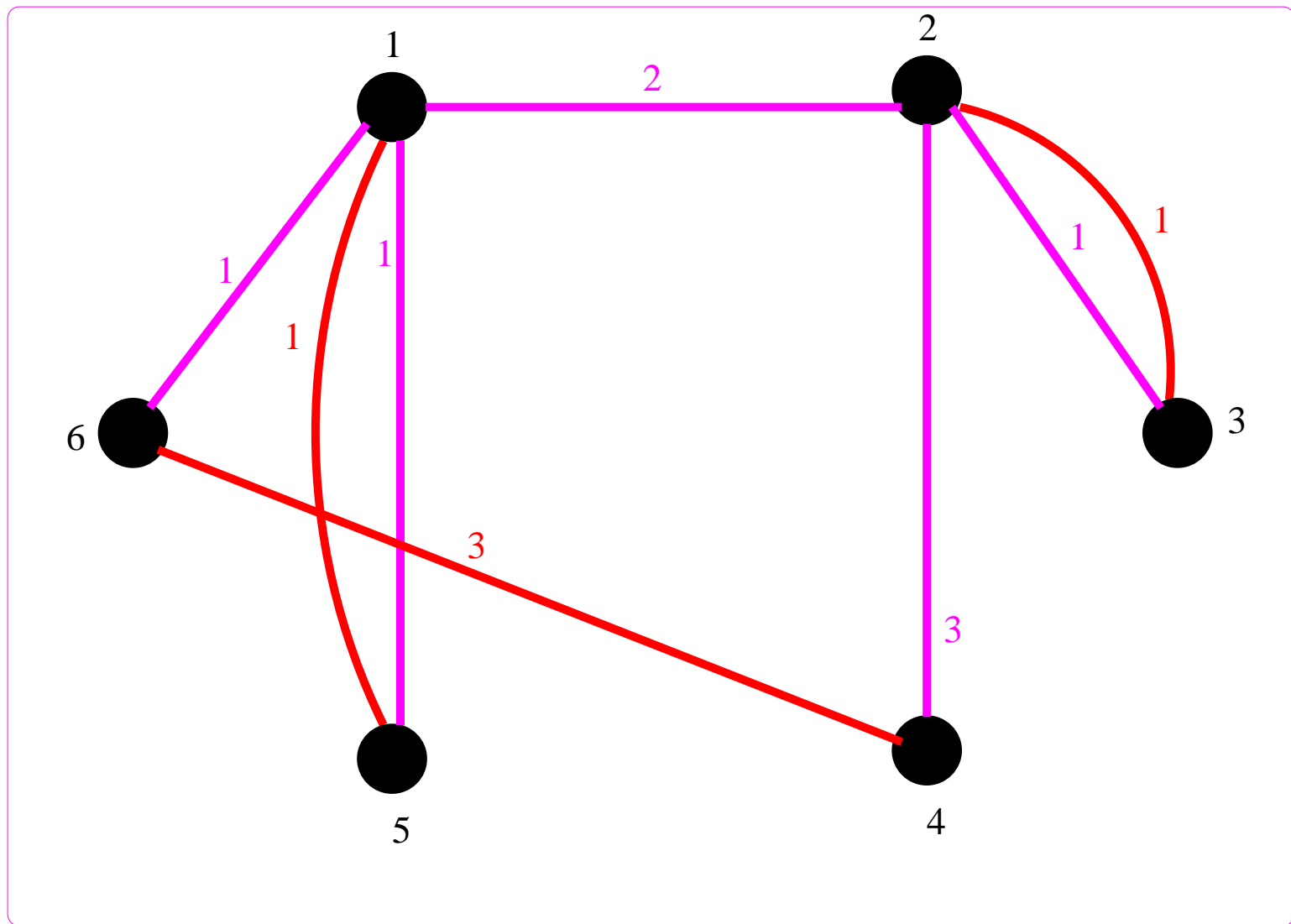
$$W + W_1 \leq W_0 + \frac{1}{2}W_0 = \frac{3}{2}W_0,$$

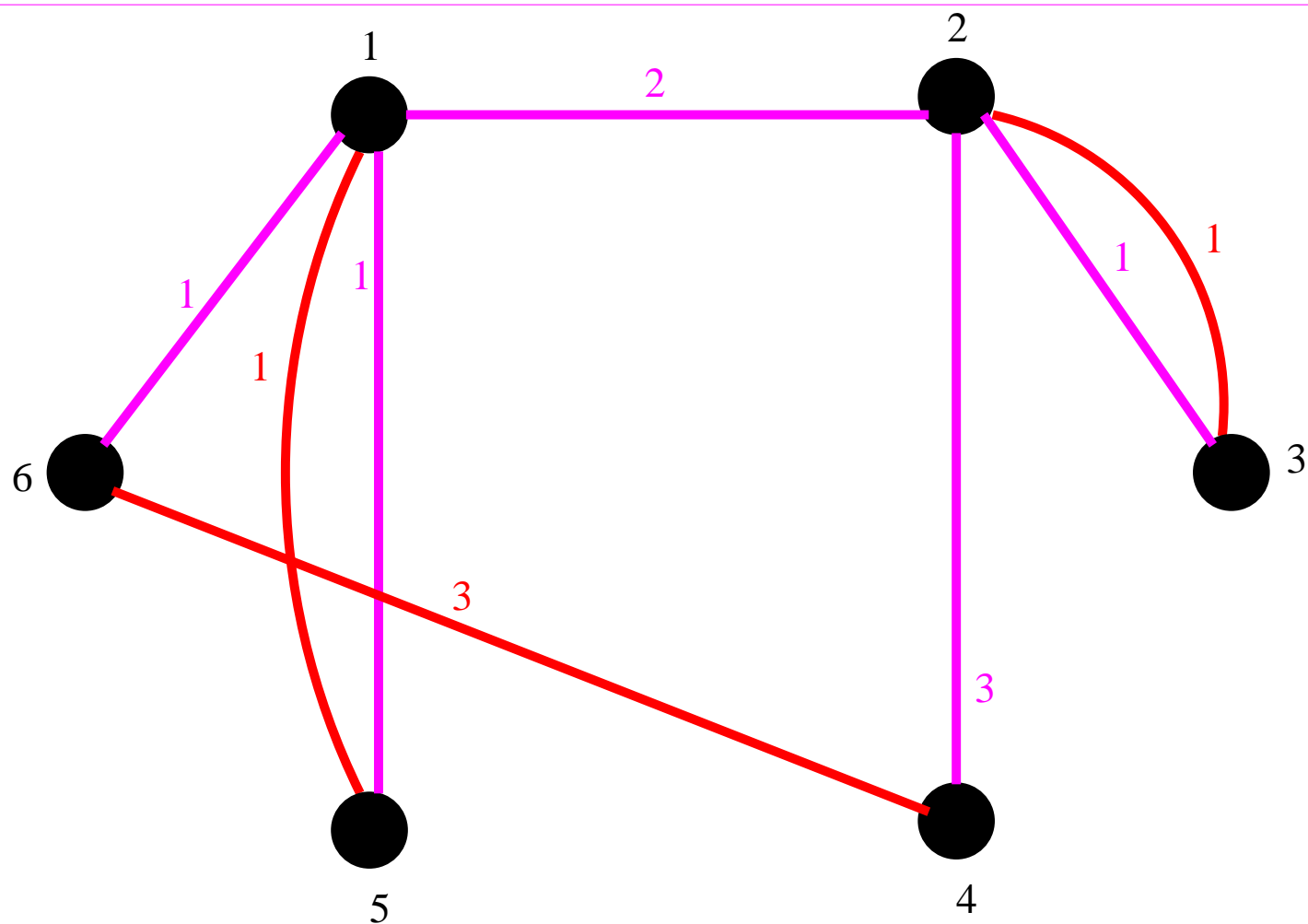
co kończy dowód twierdzenia. ■











$C_0 = (1, 5, 1, 2, 3, 2, 4, 6, 1)$

