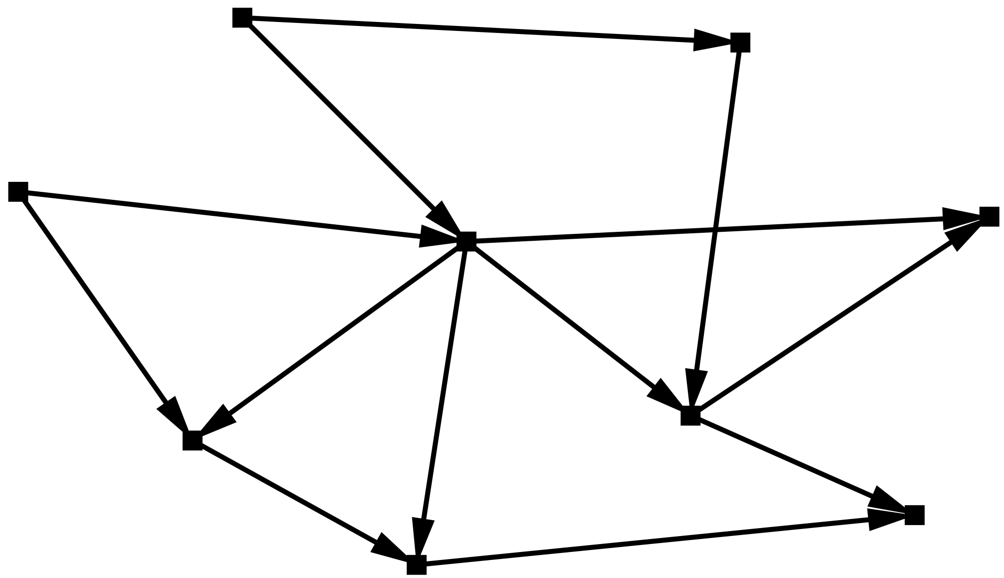
1. ja 2. detsembril on loeng ja praktikum ära vahetatud

Suunatud graaf $G = (V, E)$ on *suunatud tsükliteta*, kui iga kahe tipu $u, v \in V$ jaoks ei leidu teed u-st v-sse või v-st u-sse.

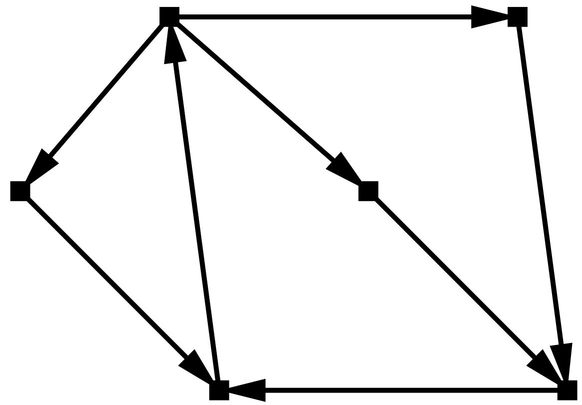
G on *tugevalt sidus*, kui iga kahe tipu $u, v \in V$ jaoks leidub tee nii u-st v-sse kui ka v-st u-sse.

Suunatud graafi G *tugevalt sidusad komponendid* on tema maksimaalsed alamgraafid, mis on tugevalt sidusad.

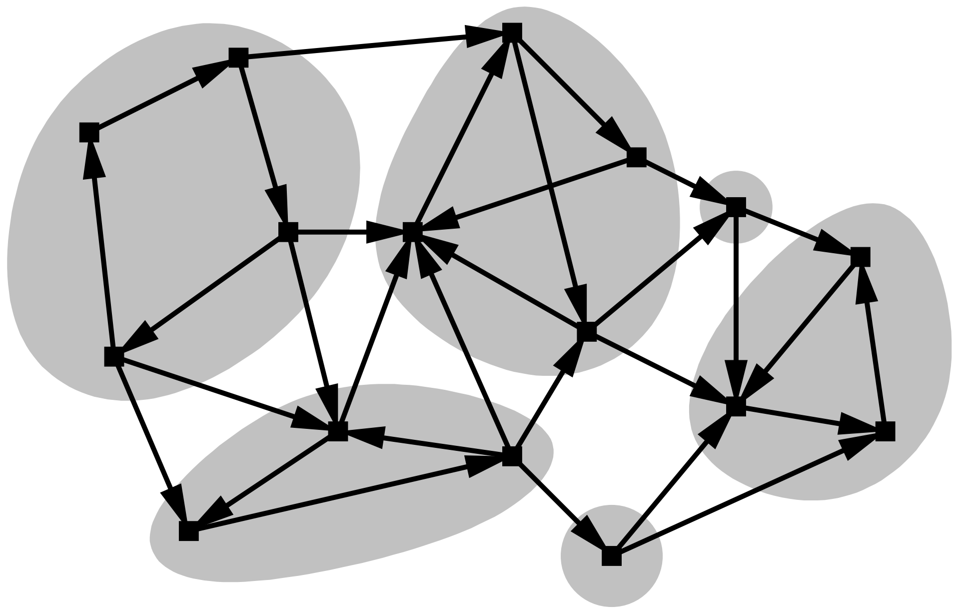
- Maksimaalne tipuhulkade sisalduvuse mõttes.



Suunatud tsükliteta graaf.



Tugevalt sidus graaf.

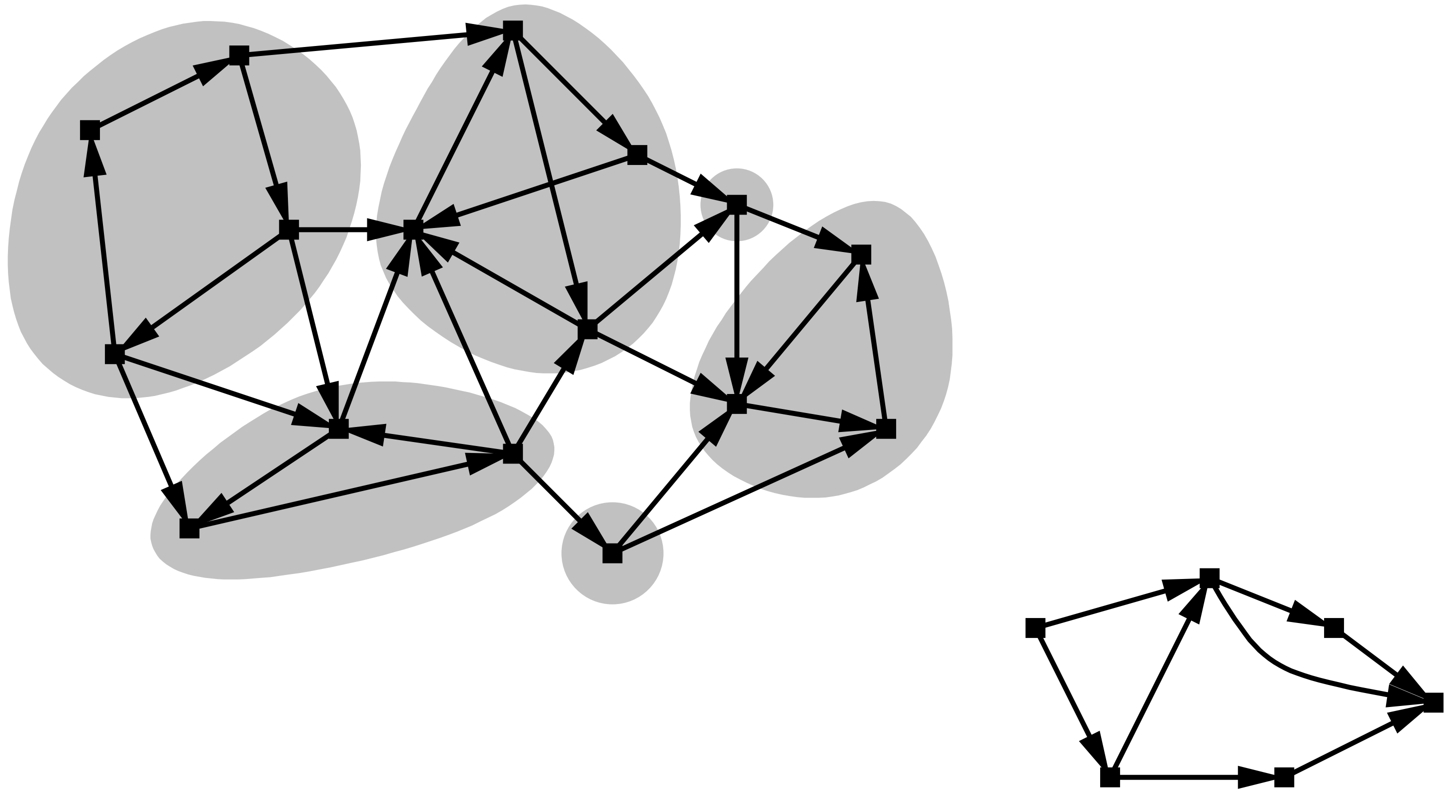


Graafi tugevalt sidusad komponendid.

Graafi $G = (V, E)$ *komponentgraafiks* nimetame graafi G^{komp}, mille

- tippudeks on graafi G tugevalt sidusad komponendid;
- kaar graafi G^{komp} tipust V_i tippu V_j (siin $V_i, V_j \subseteq V$) leidub parajasti siis, kui G-s leidub serv mõnest V_i-sse kuuluvast tipust mõnda V_j-i kuuluvasse tippu.

Graaf G^{komp} on suunatud tsükliteta.

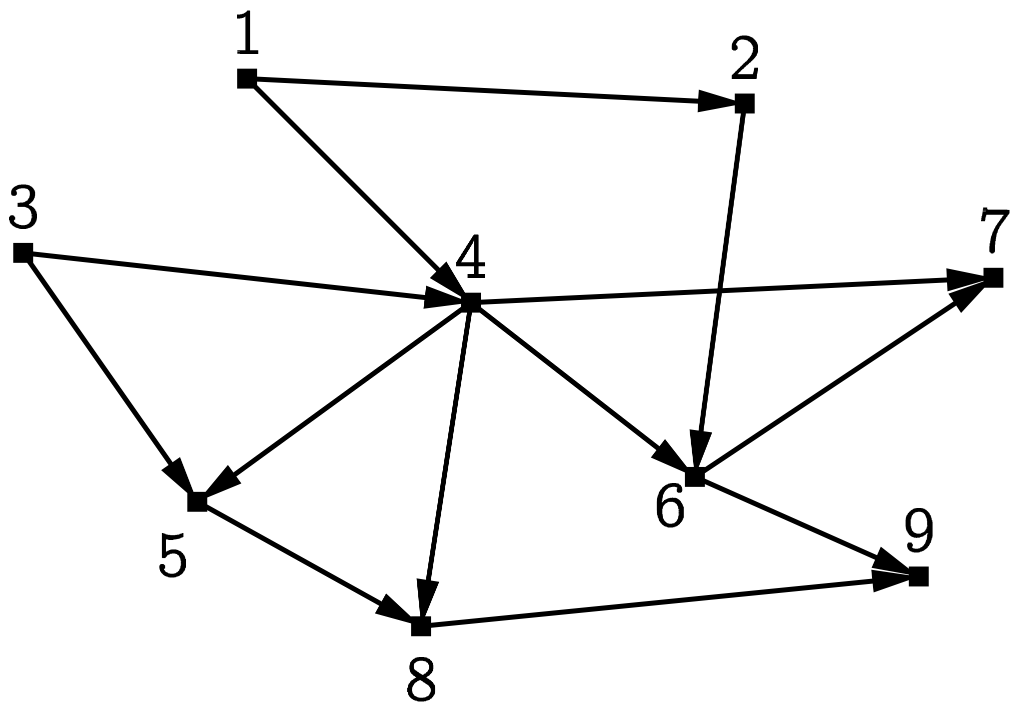


Näide komponentgraafist.

Kui graaf on suunatud tsükliteta, siis saab tema tippe selliselt järjestada, et igast tipust lähevad servad ainult selles järjestuses kaugemal olevatesse tippudesse.

Kui selline järjestus on antud, siis ütleme, et tipud on *topoloogiliselt sorteeritud*.

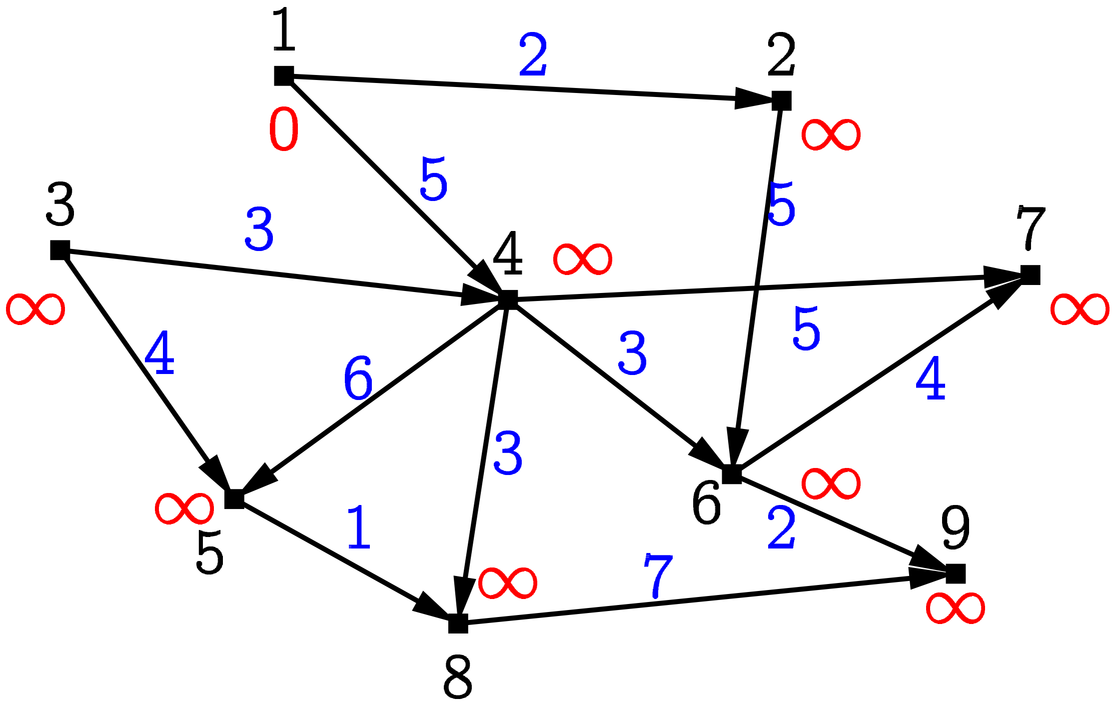
Algoritm tippude topoloogiliselt sorteerimiseks: Kiho konspekt, joonis 6.2. Tema keerukus on $O(|E|)$.

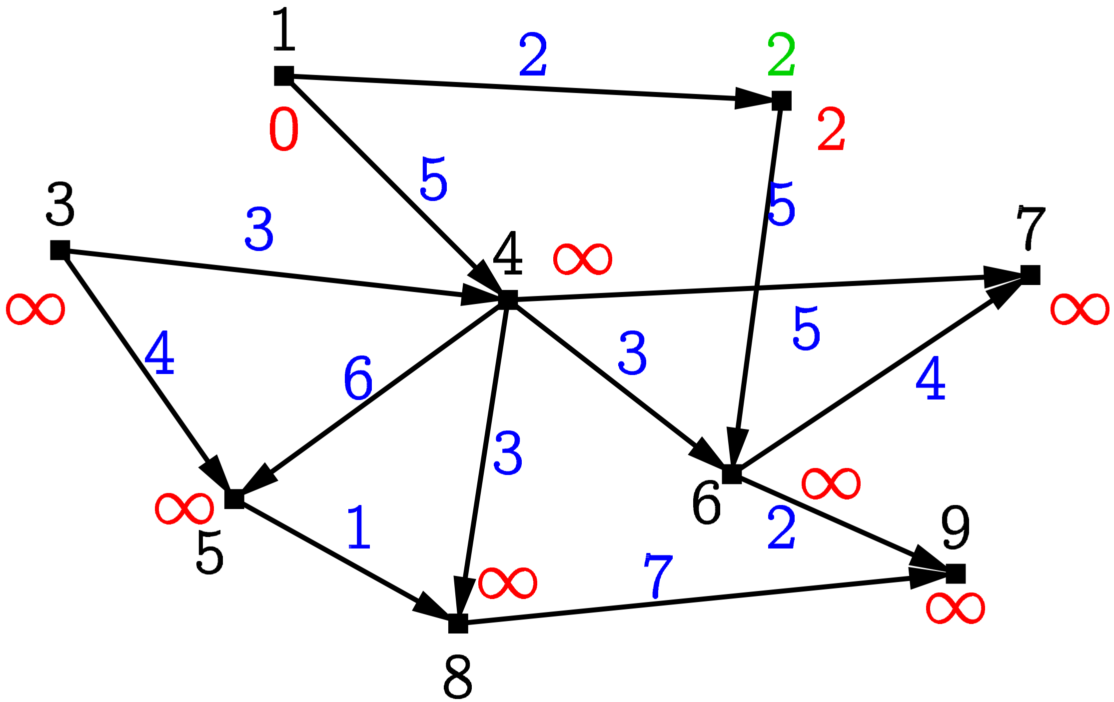


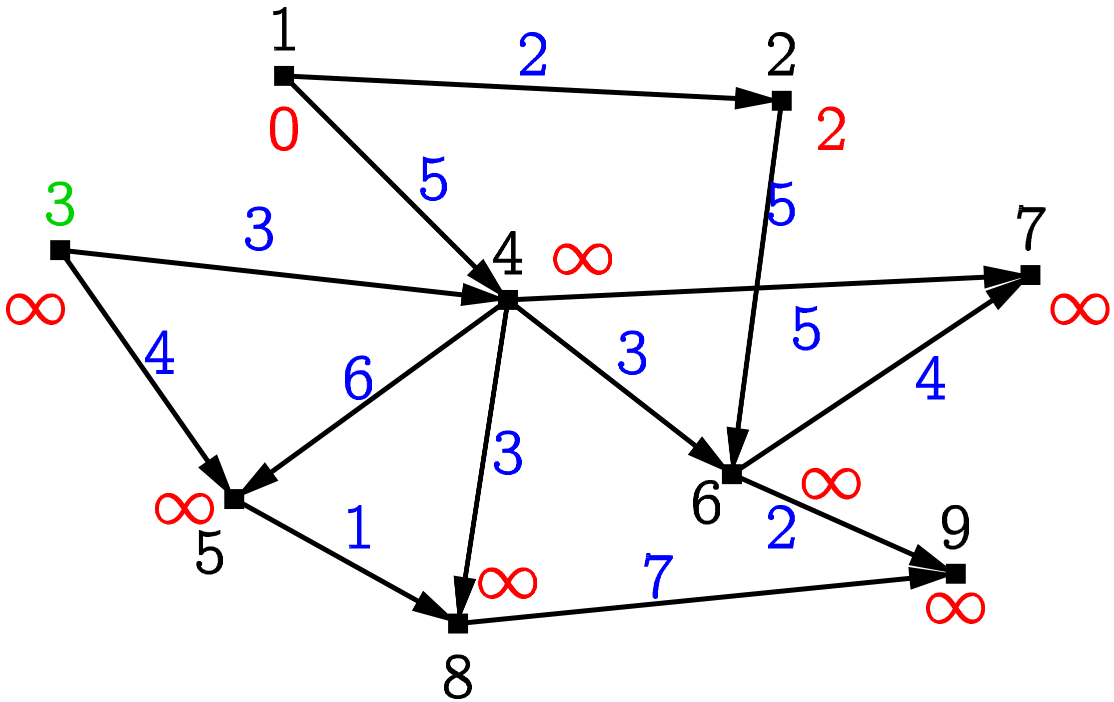
Näide suunatud tsükliteta graafi tippude topoloogilisest sorteerimisest.

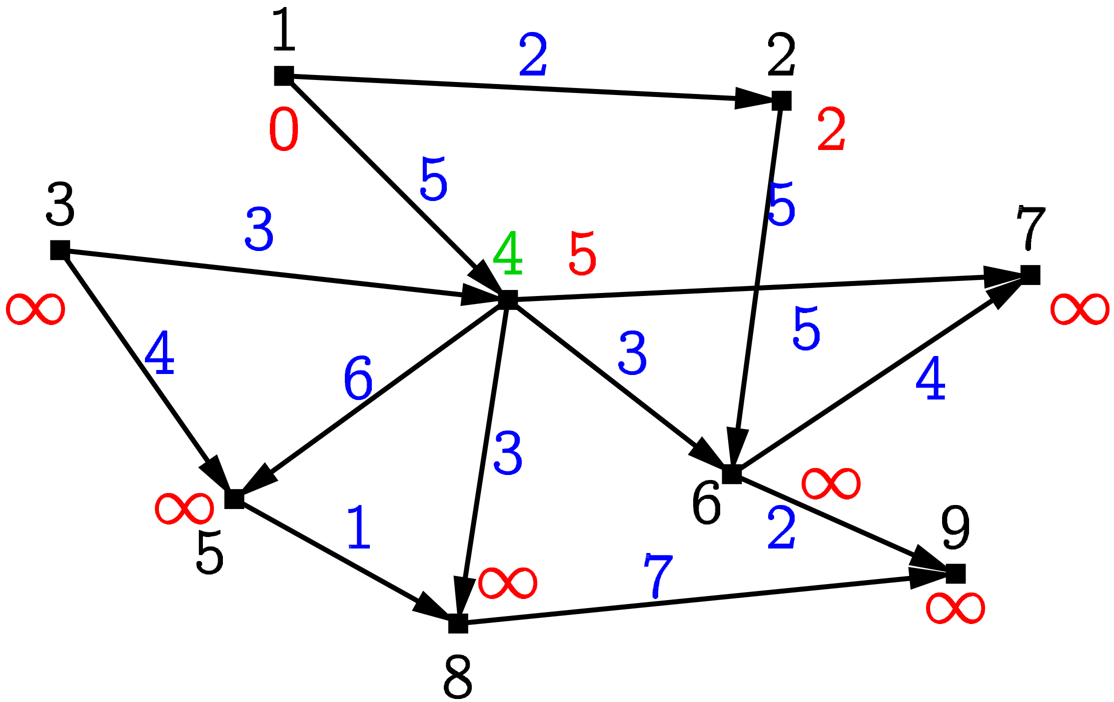
Suunatud tsükliteta graafis saab lühimaid teid mingist fikseeritud tipust u kõigisse teistesse tippudesse leida ajaga $O(|E|)$.

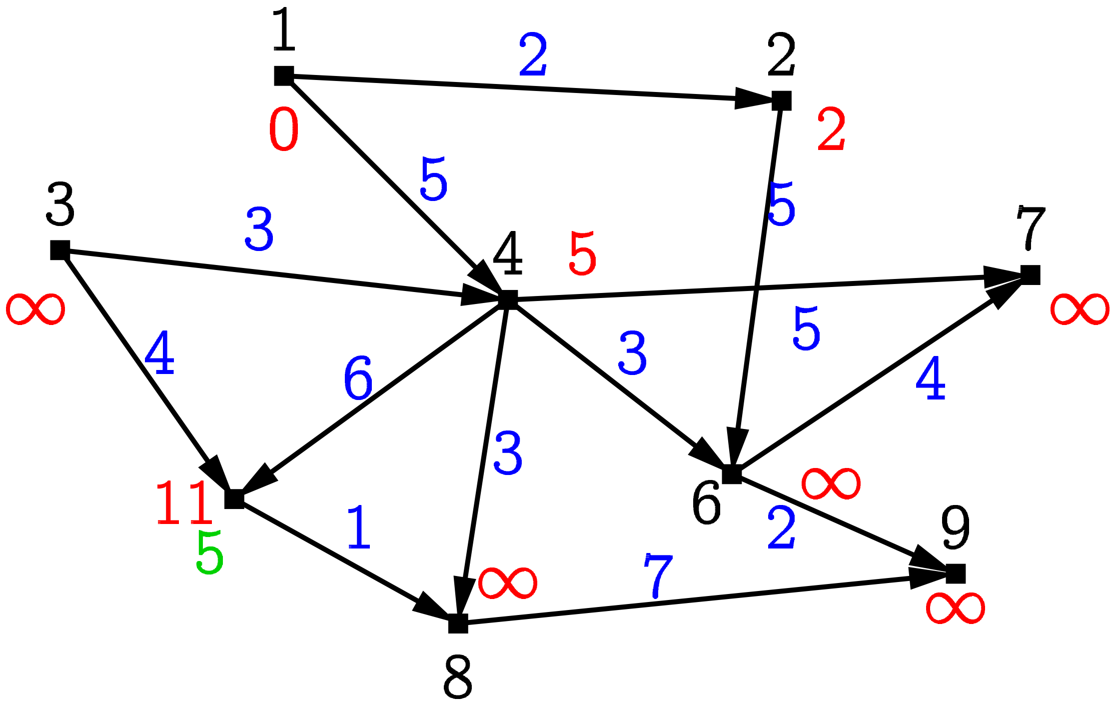
$\{v_1, \ldots, v_n\} :=$ topol_sorteeri(G)
for all $v \in V$ **do** $D[v] := \infty$
$D[u] := 0$
for $i := 1$ **to** n **do**
    for all $w \in G^{-1}v_i$ **do**
        $D[v_i] := \min(D[v_i], D[w] + \ell(w, v_i))$
return D


1
0
2
2
∞
5
5
3
3
∞
4
∞
7
∞
5
4
6
3
3
4
∞
6
∞
1
5
2
9
3
∞
7
∞
8

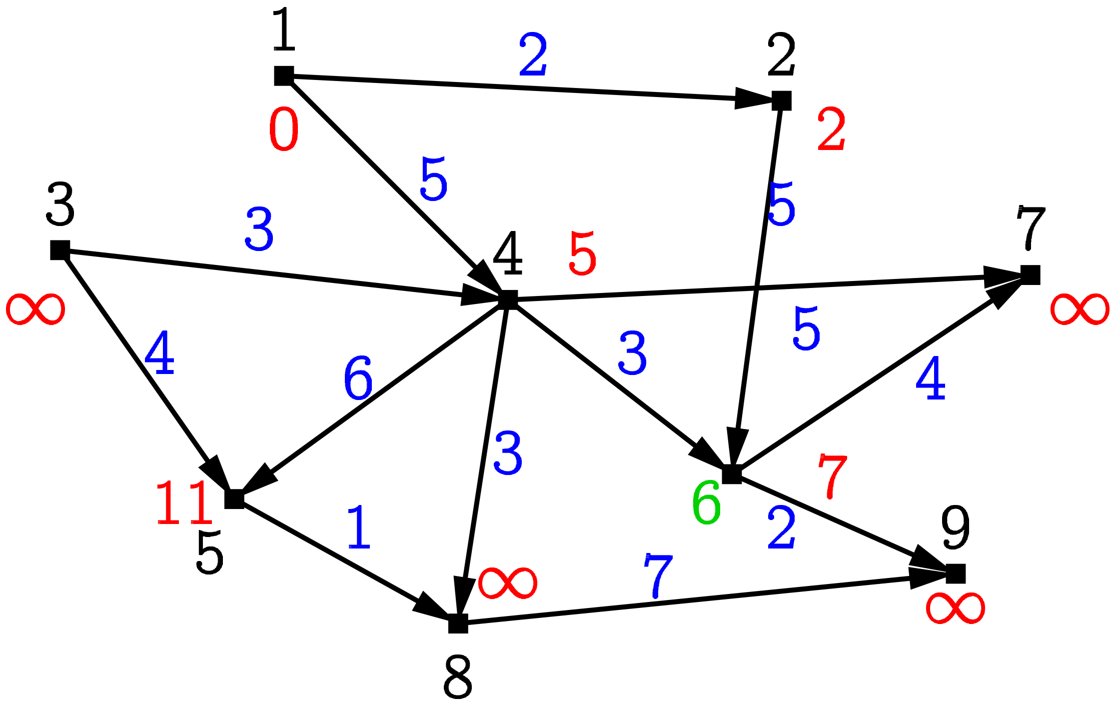

1
2
2
0
2
5
3
3
5
4
∞
7
∞
∞
5
4
6
3
3
4
∞
6
2
9
∞
5
1
∞
7
∞
8

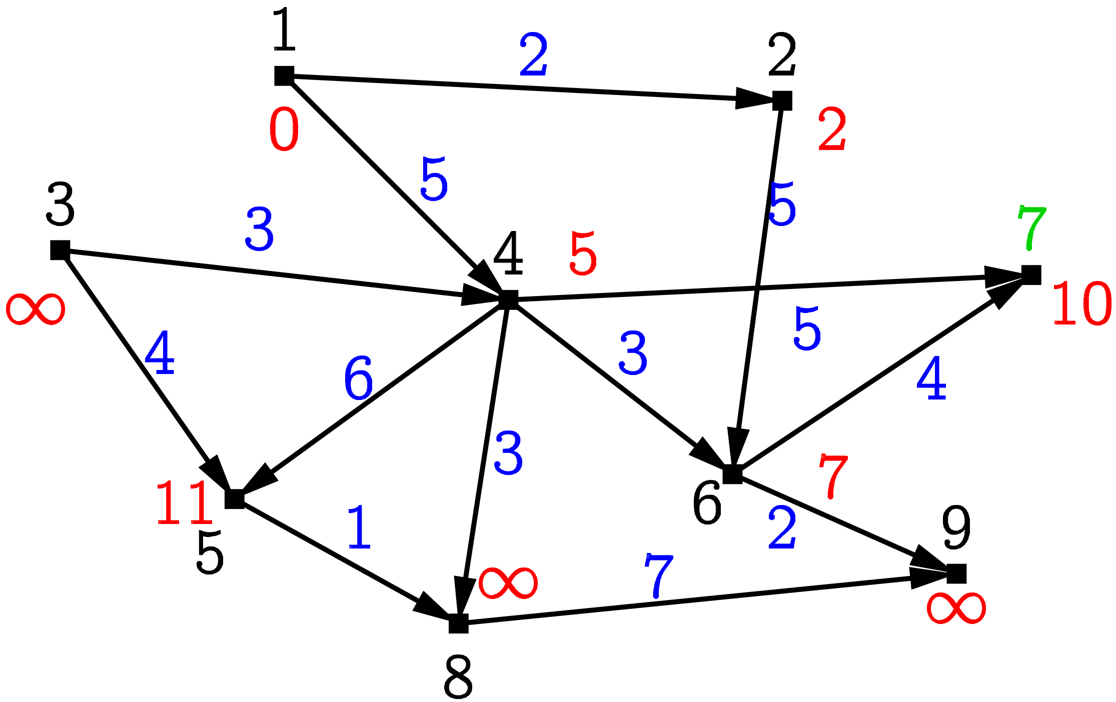

1
0
2
2
2
5
3
3
∞
3
5
4
∞
7
∞
5
4
6
3
3
4
∞
∞
5
1
6
∞
2
9
8
7
∞
∞

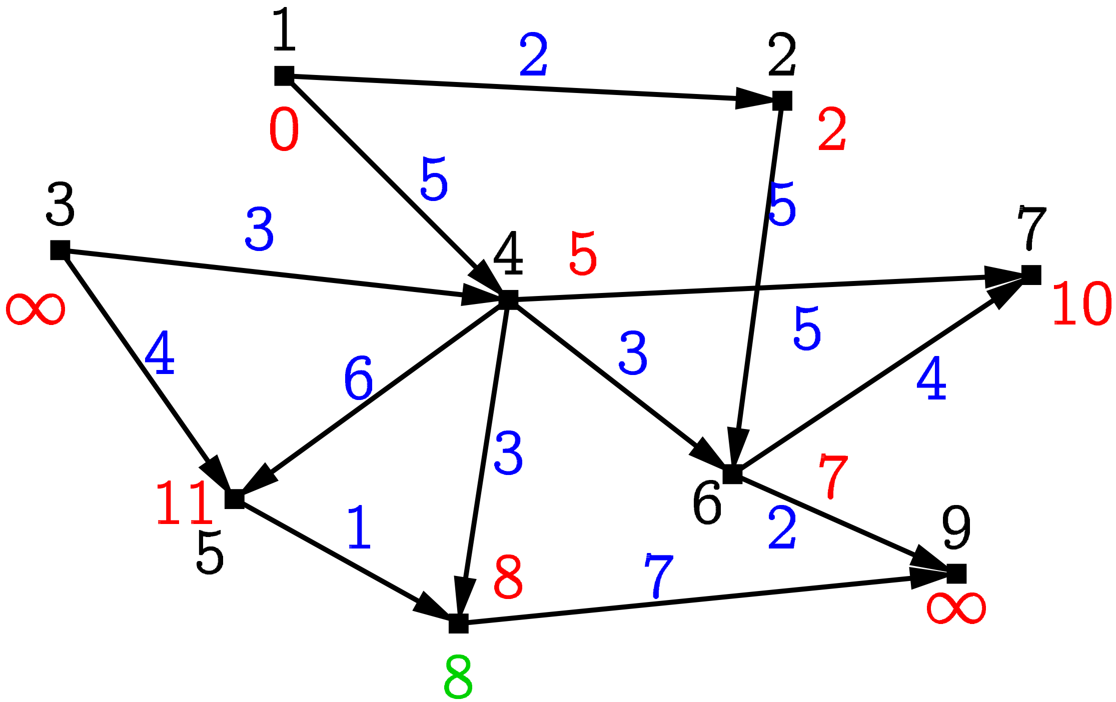

1
0
2
2
2
5
3
3
∞
5
5
4
5
7
∞
4
6
3
3
5
4
∞
6
∞
5
1
3
2
9
∞
7
8
∞

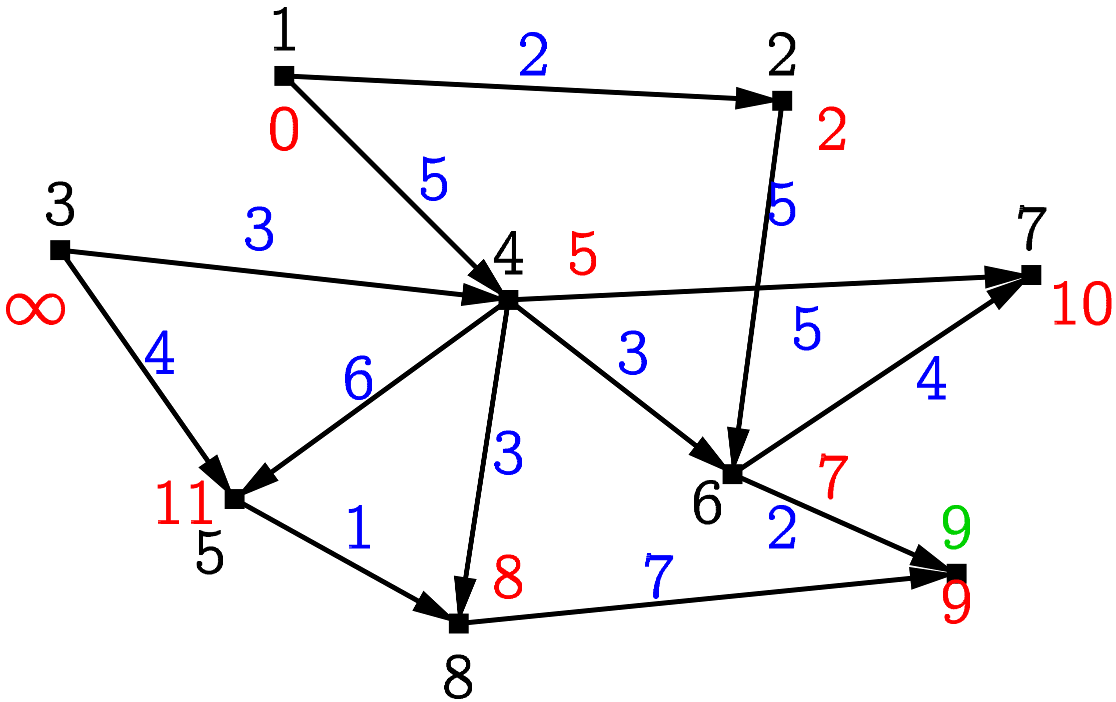


1
2
0
2
5
3
3
5
4
5
7
5
∞
4
6
3
3
5
4
∞
11
5
6
∞
2
9
1
7
∞
∞
8


1
2
0
2
2
5
5
3
3
5
4
7
∞
∞
4
3
5
6
3
5
4
11
6
7
1
9
5
2
∞
7
∞
8


1
0
2
2
2
5
5
3
3
∞
4
5
7
10
5
4
4
6
3
3
5
6
7
11
5
1
2
9
∞
7
8
∞


1
2
0
2
3
5
5
4
5
7
∞
3
10
5
4
6
3
4
3
7
6
11
1
2
9
5
8
7
∞
8


1
2
0
2
2
5
3
5
3
7
4
5
∞
10
5
4
6
3
3
4
6
7
11
9
1
2
5
8
7
9
8

Tsükkel ridades 4–5 toimub üle kõigi servade. Sellest ka keerukus $O(|E|)$.

Seda tsüklit võib teha suvalises sellises järjekorras, et kui serv (w_1, w_2) vaadatakse läbi enne serva (w_3, w_4), siis $w_1 \leqslant w_4$.

Siis võib kindel olla, et (w_3, w_4) läbivaatus ei mõjuta w_1 kaugust u-st.

Muuhulgas siis ka...

4 **for** $i := 1$ **to** n **do**
5   **for all** $w \in Gv_i$ **do**
6         $D[w] := \min(D[w], D[v_i] + \ell(v_i, w))$

Kuidas leida graafi tugevalt sidusaid komponente?

Naiivne algoritm — leiame kõigi tippude vahelised kaugused, kontrollime kõigi paaride $u, v \in V$ jaoks, kas $d(u, v) < \infty$ ja $d(v, u) < \infty$.

Vaatame järgmist graafi sügavuti läbimise algoritmi, mis oma töö tulemusena järjestab tipud vastava tipu töötlemise *lõpetamise* järjekorras.

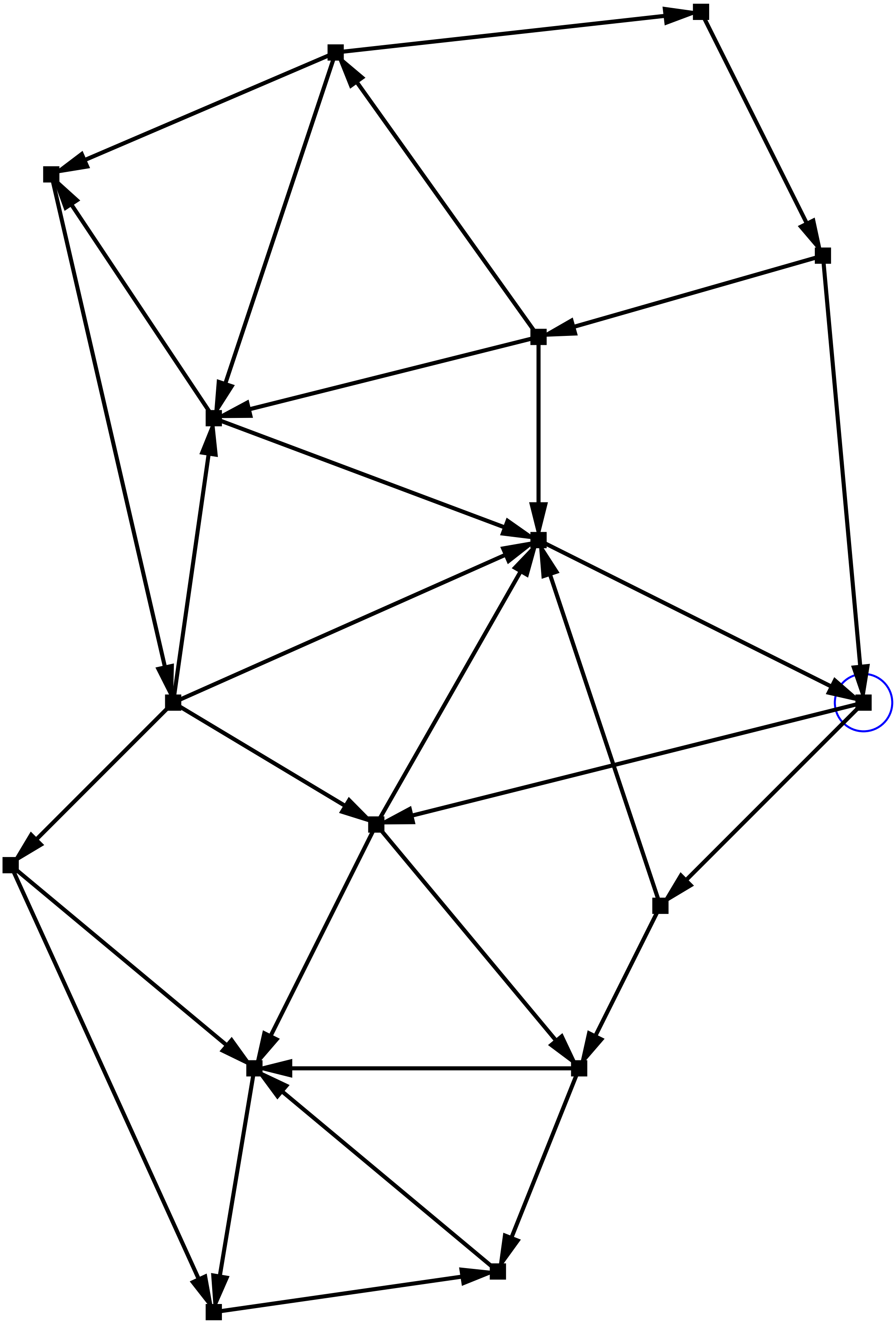
Igal tipul v olgu kaks töövälja: *v.läbitud* ja $v.jrknr$.

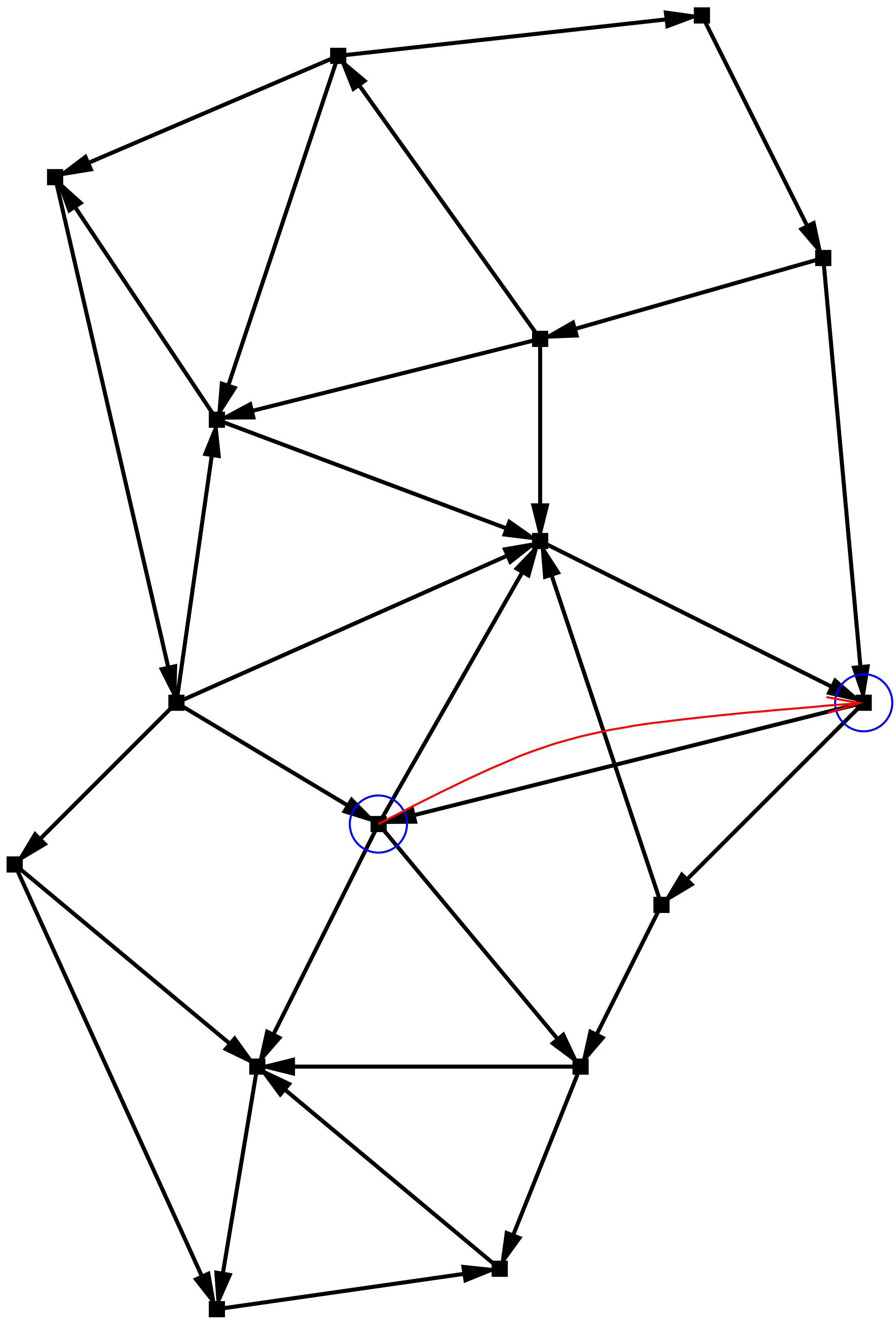
sügavuti(G) on

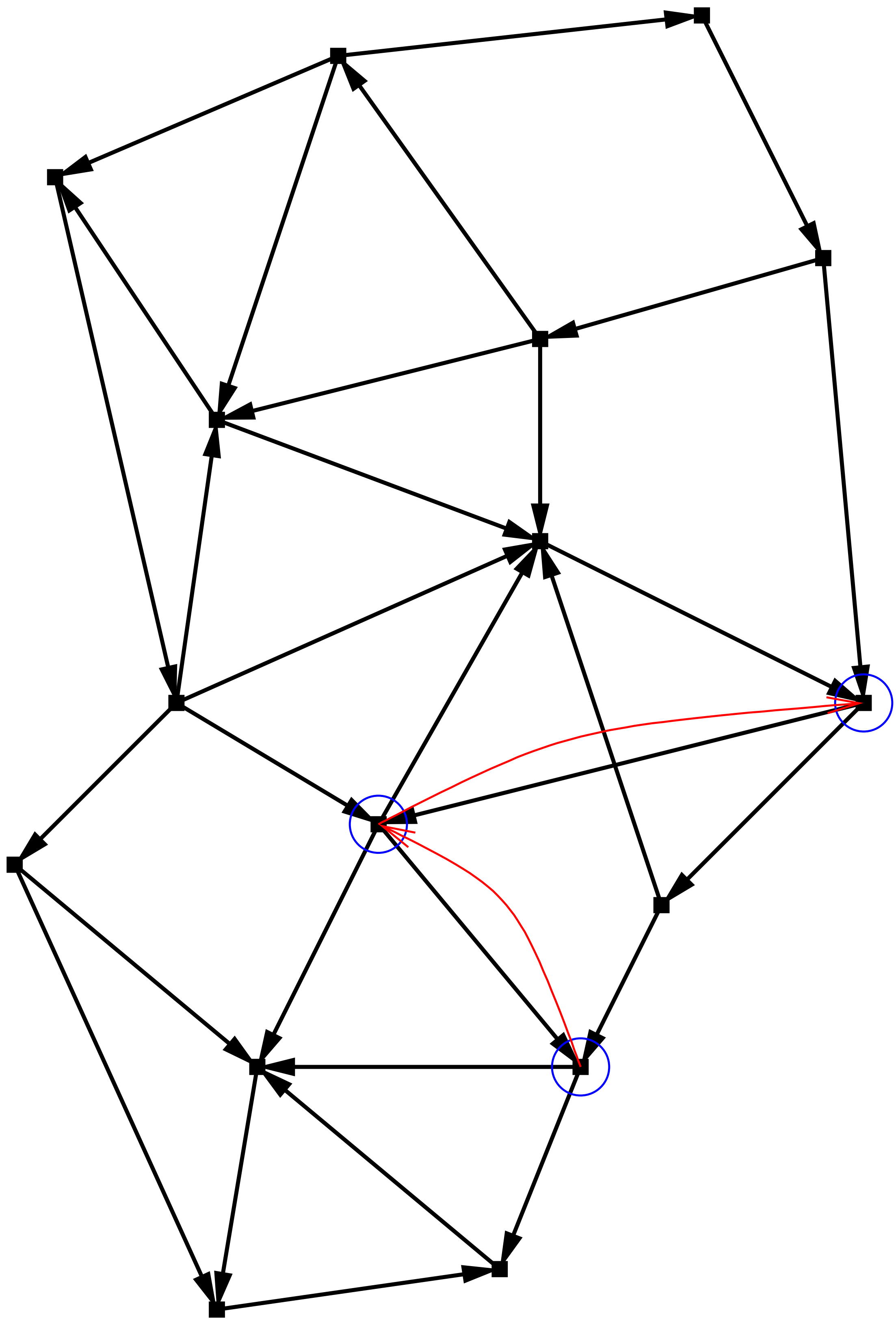
```
for all v ∈ V do v.läbitud := false
aeg := 0
for all v ∈ V do
    if ¬v.läbitud then külasta(v)
```

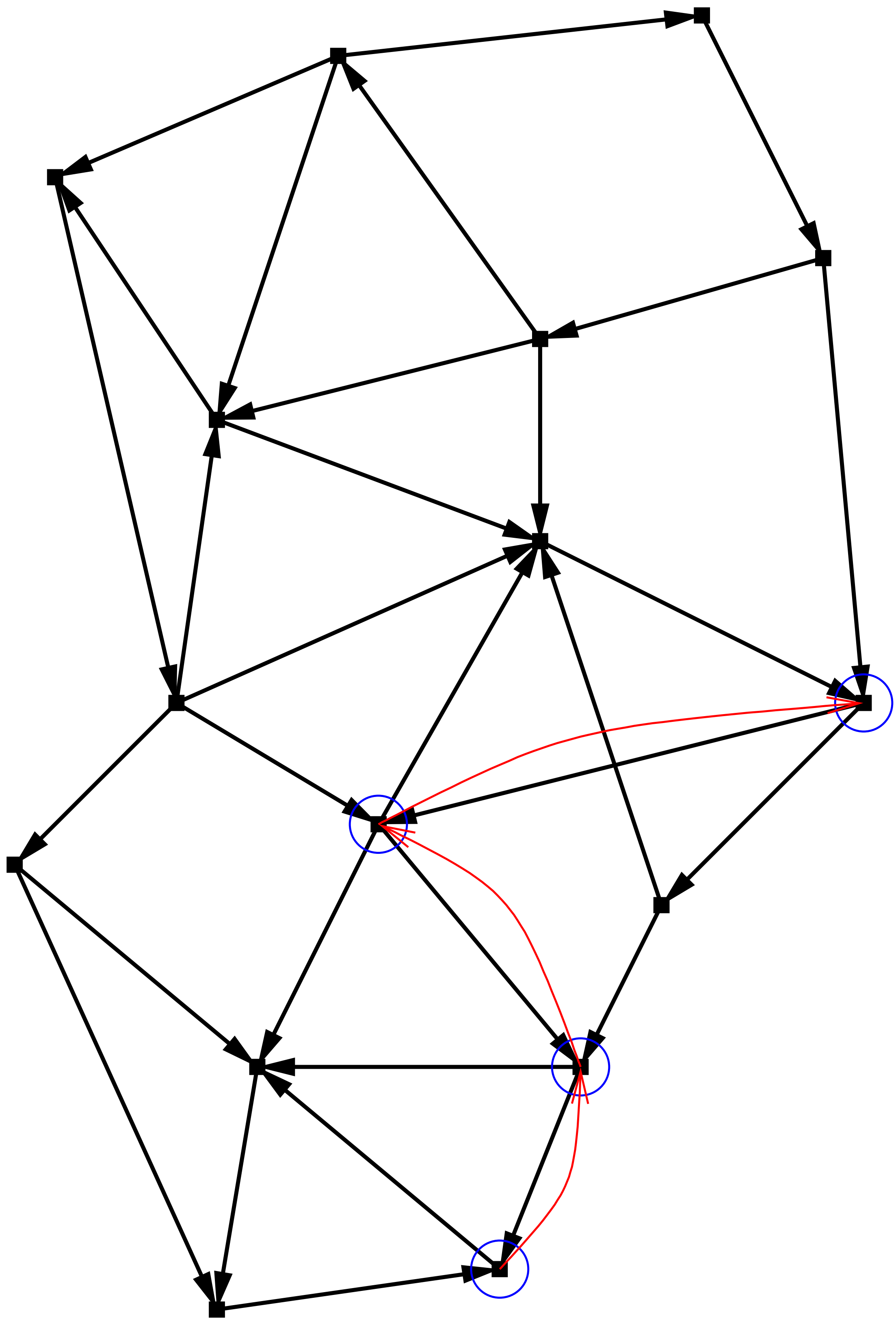
külasta(v) on

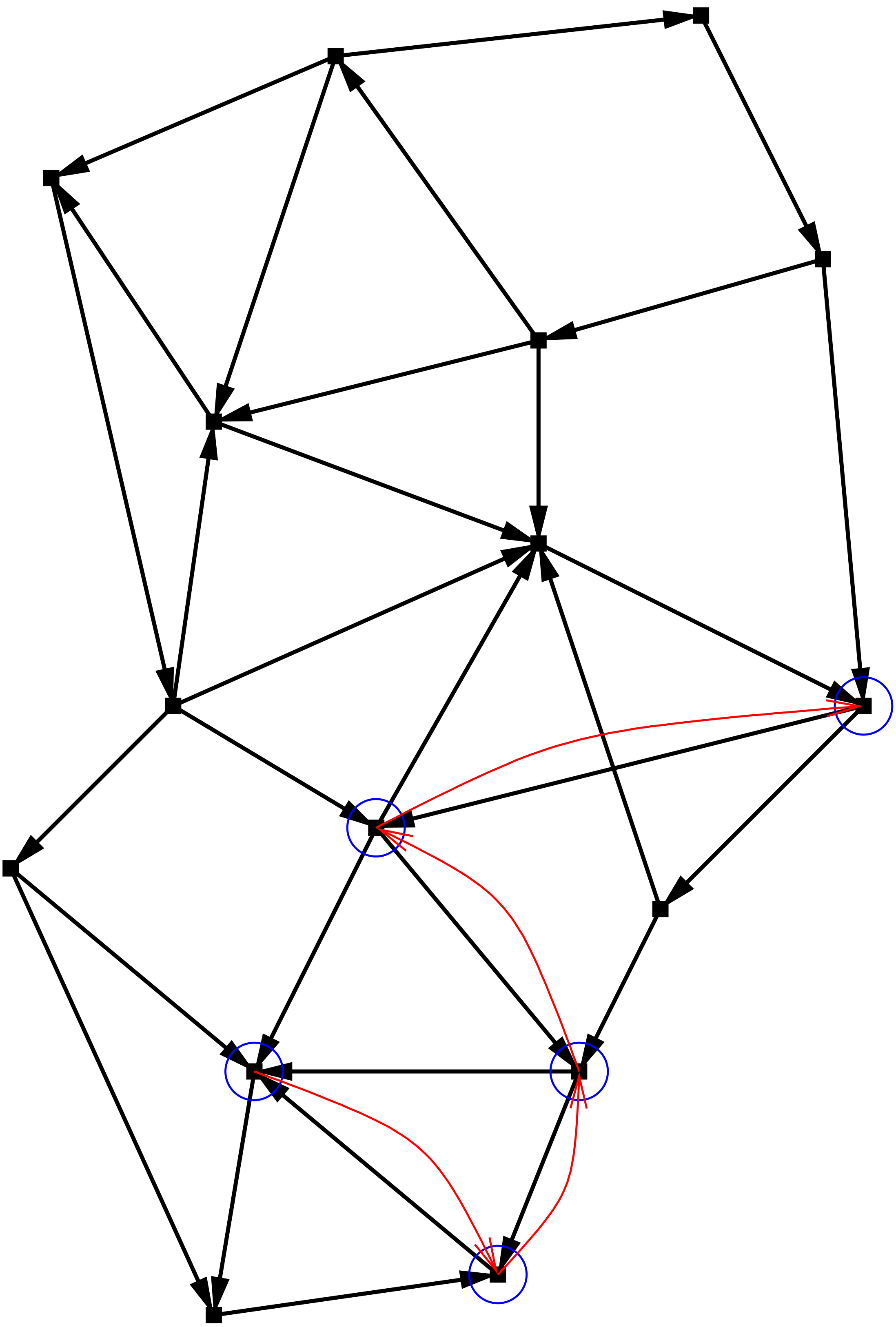
```
v.läbitud := true
for all w ∈ Gv do
    if ¬w.läbitud then külasta(w)
aeg := aeg + 1; v.jrknr := aeg
```

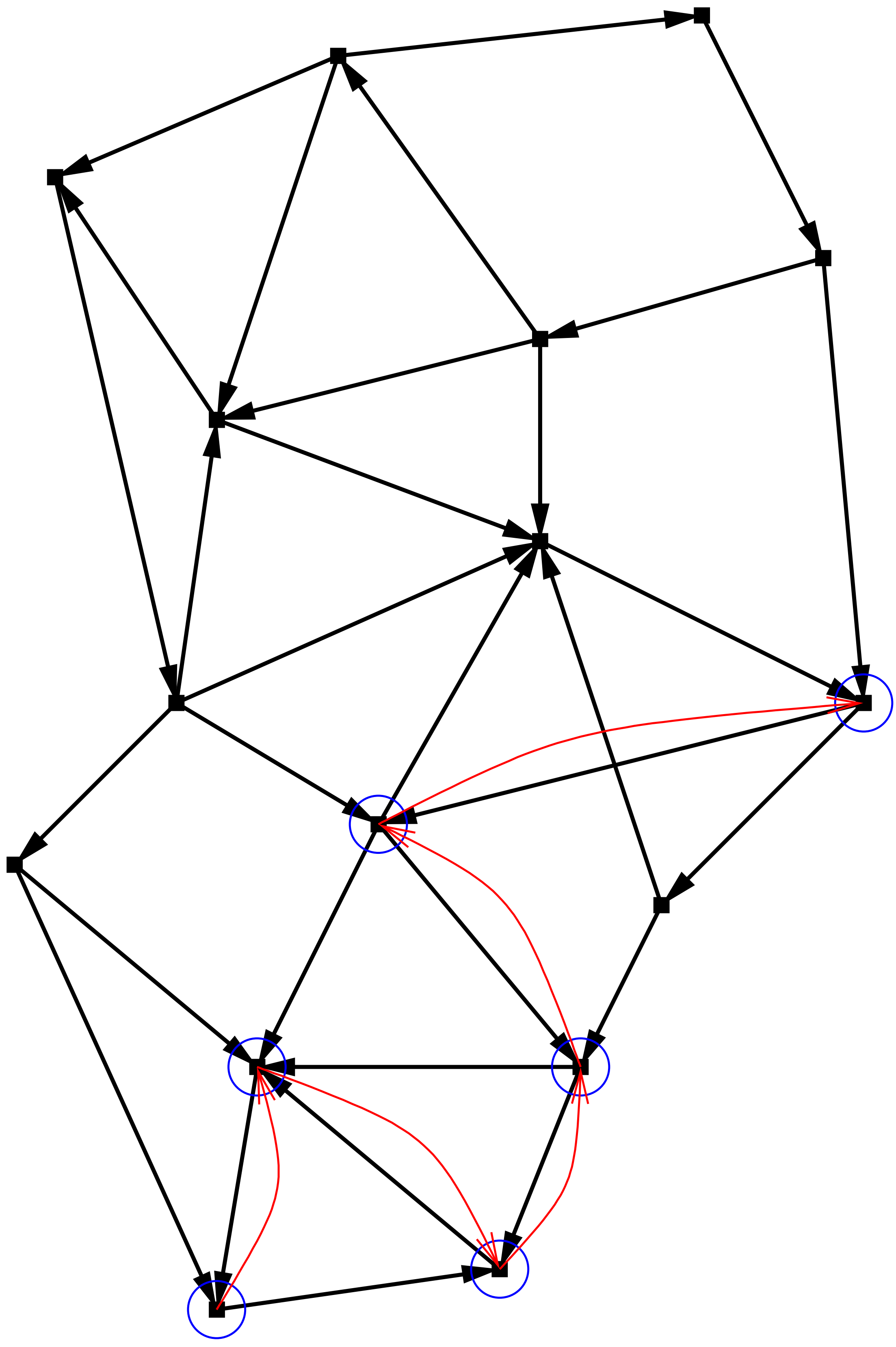


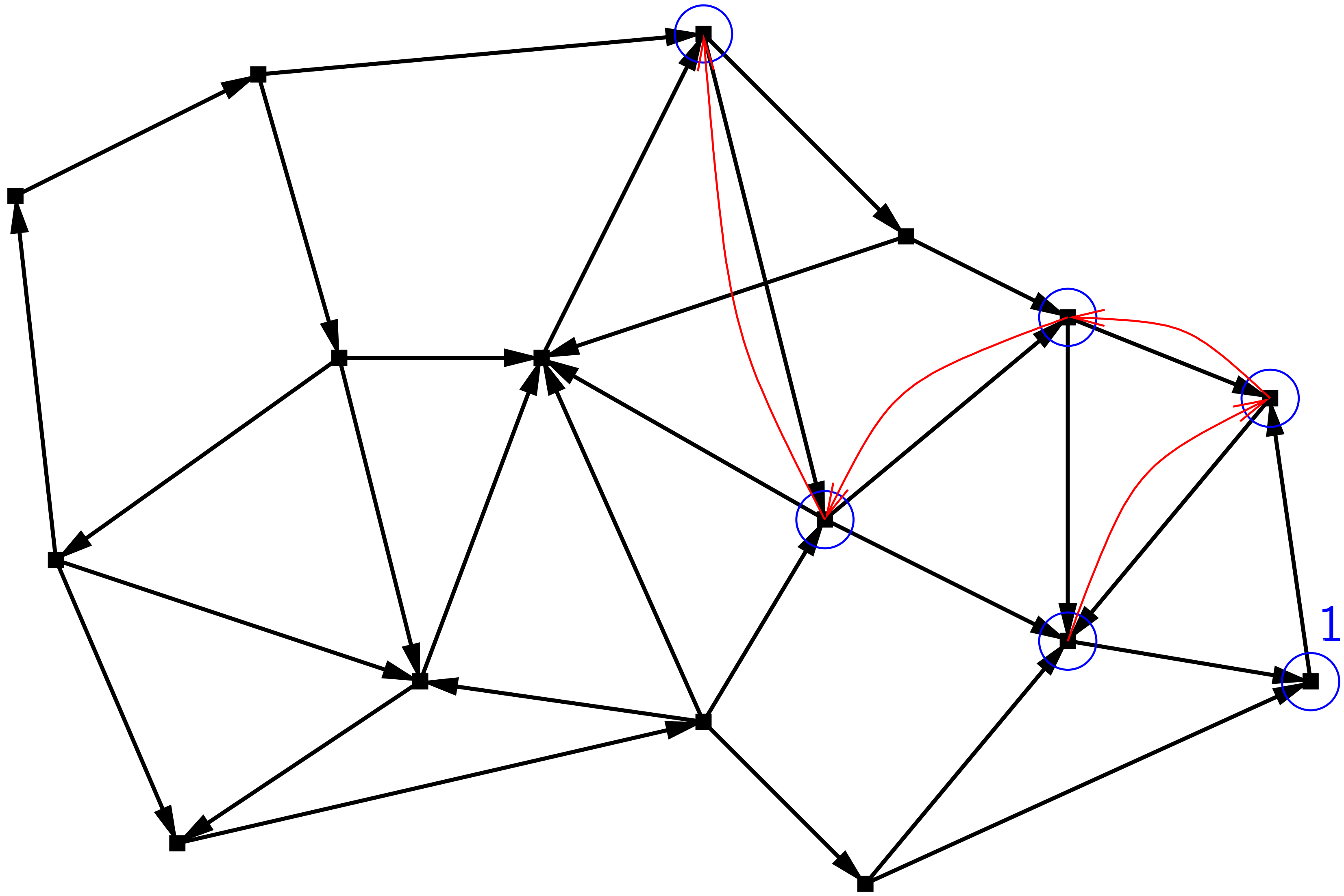


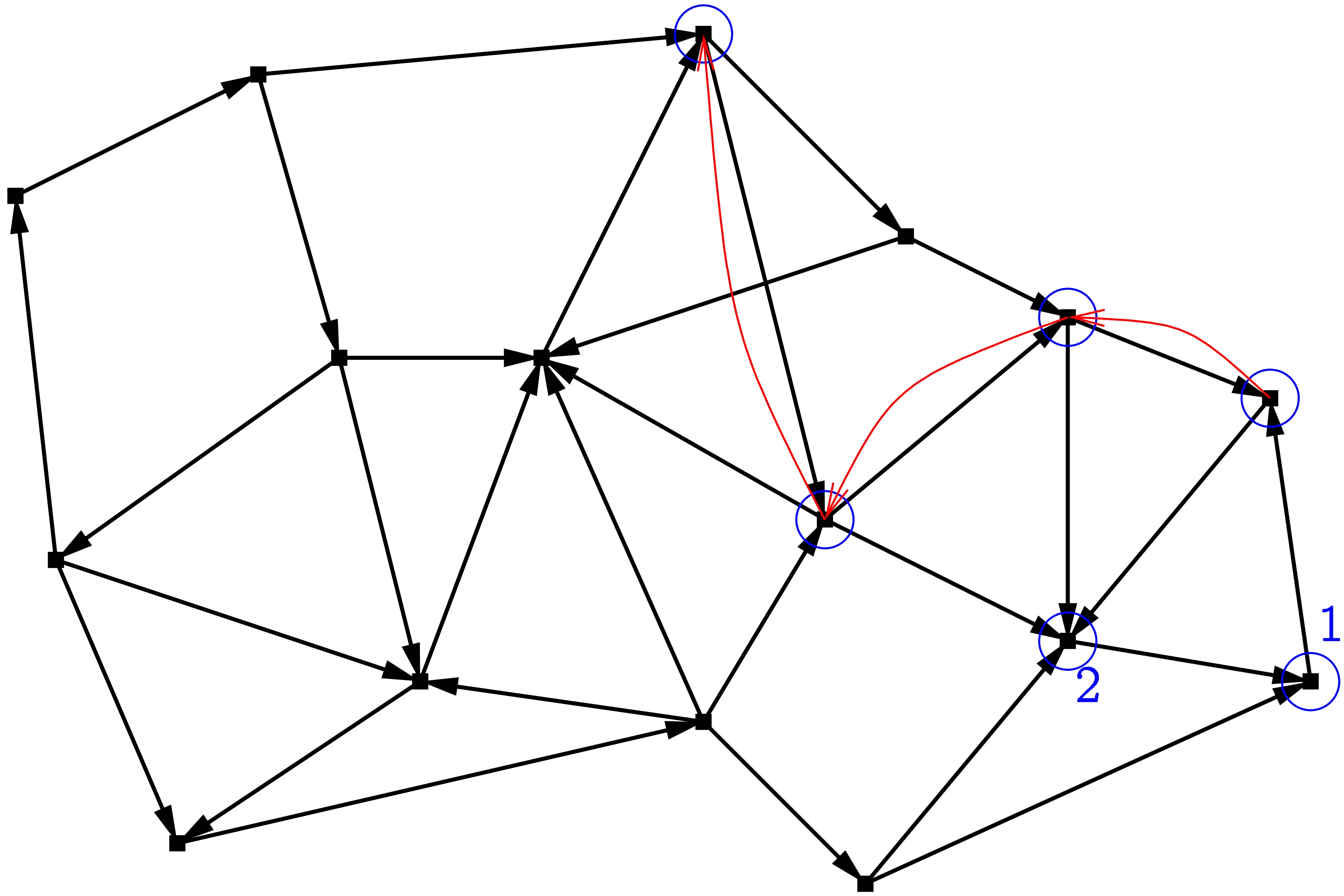


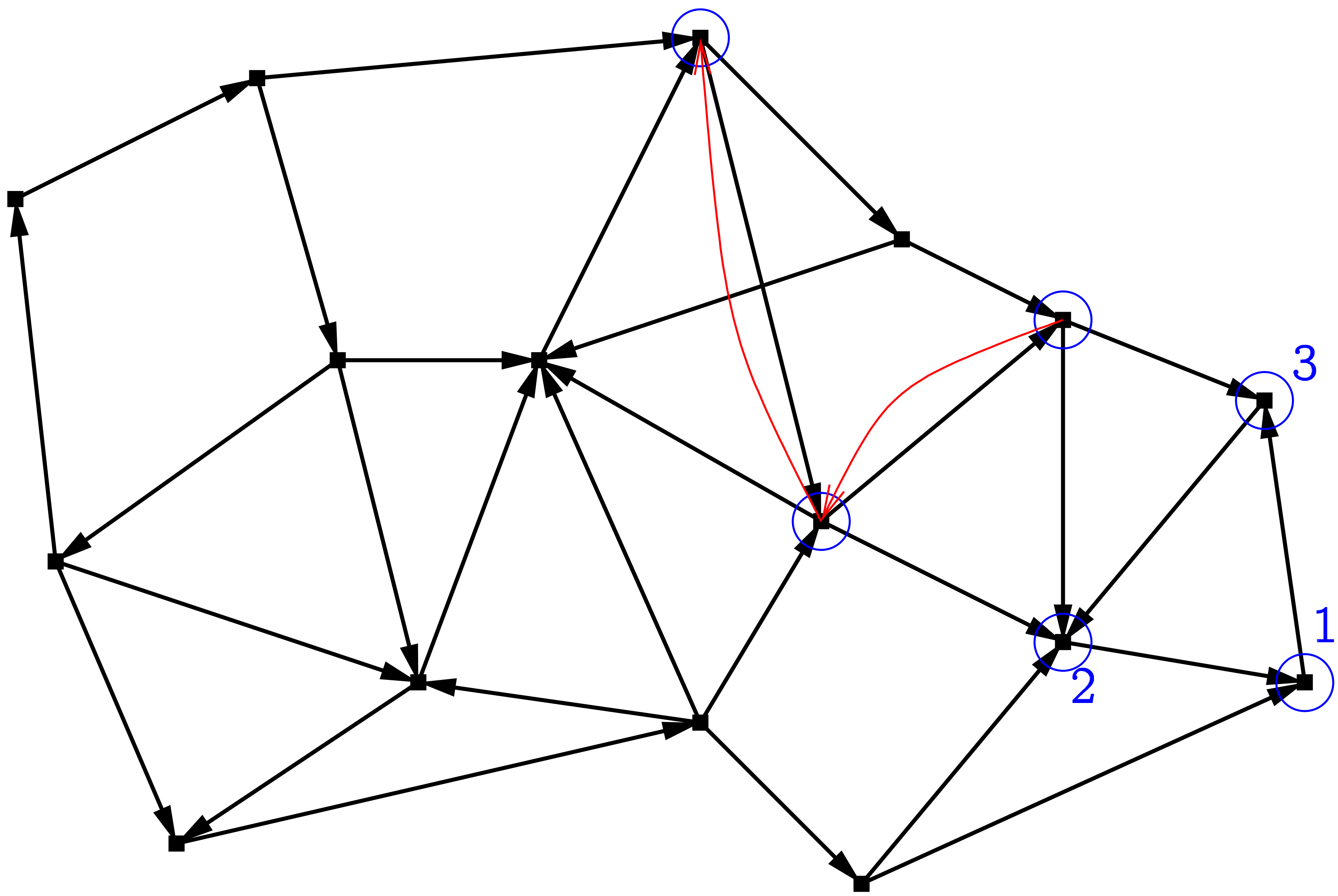


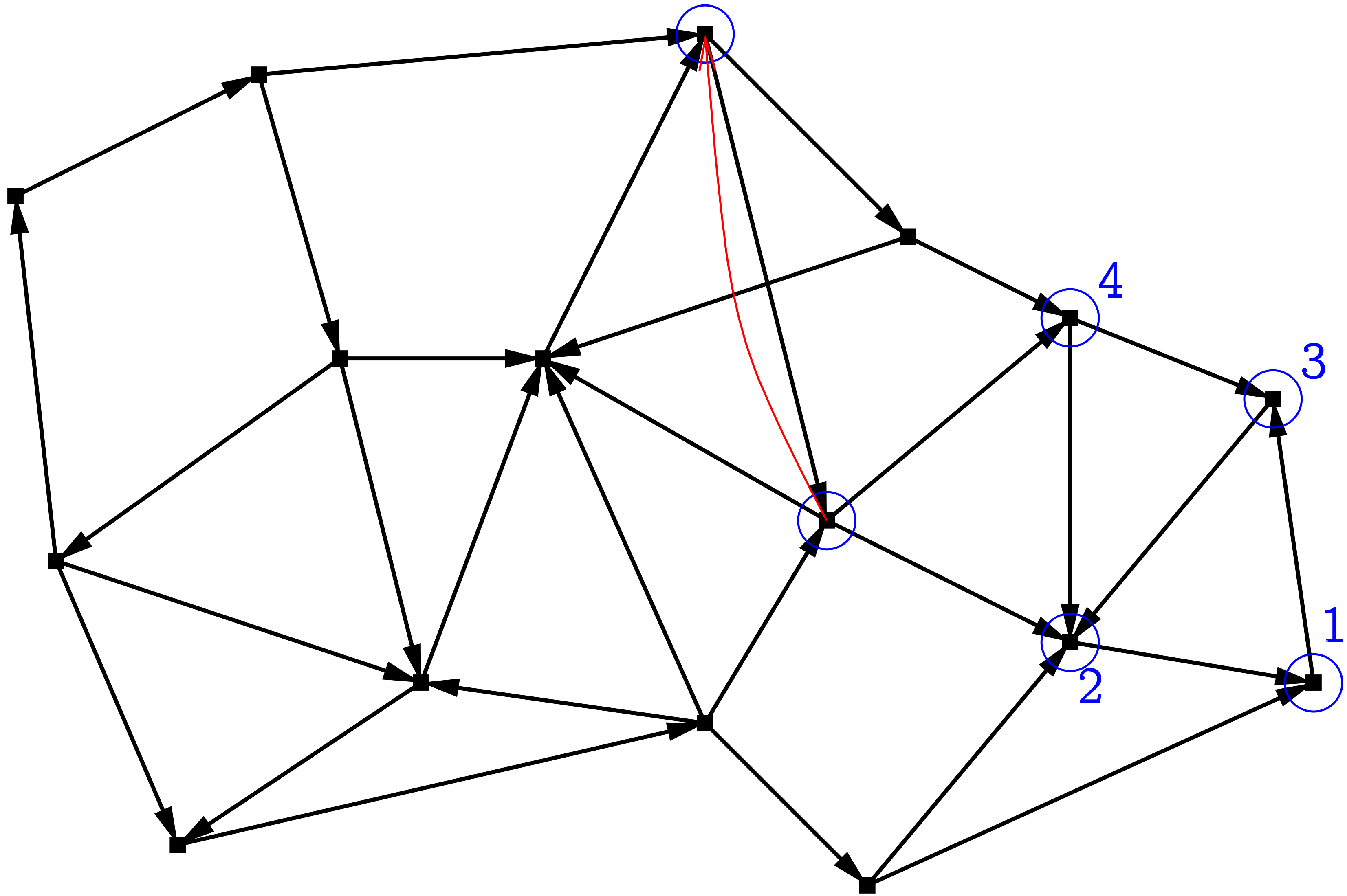


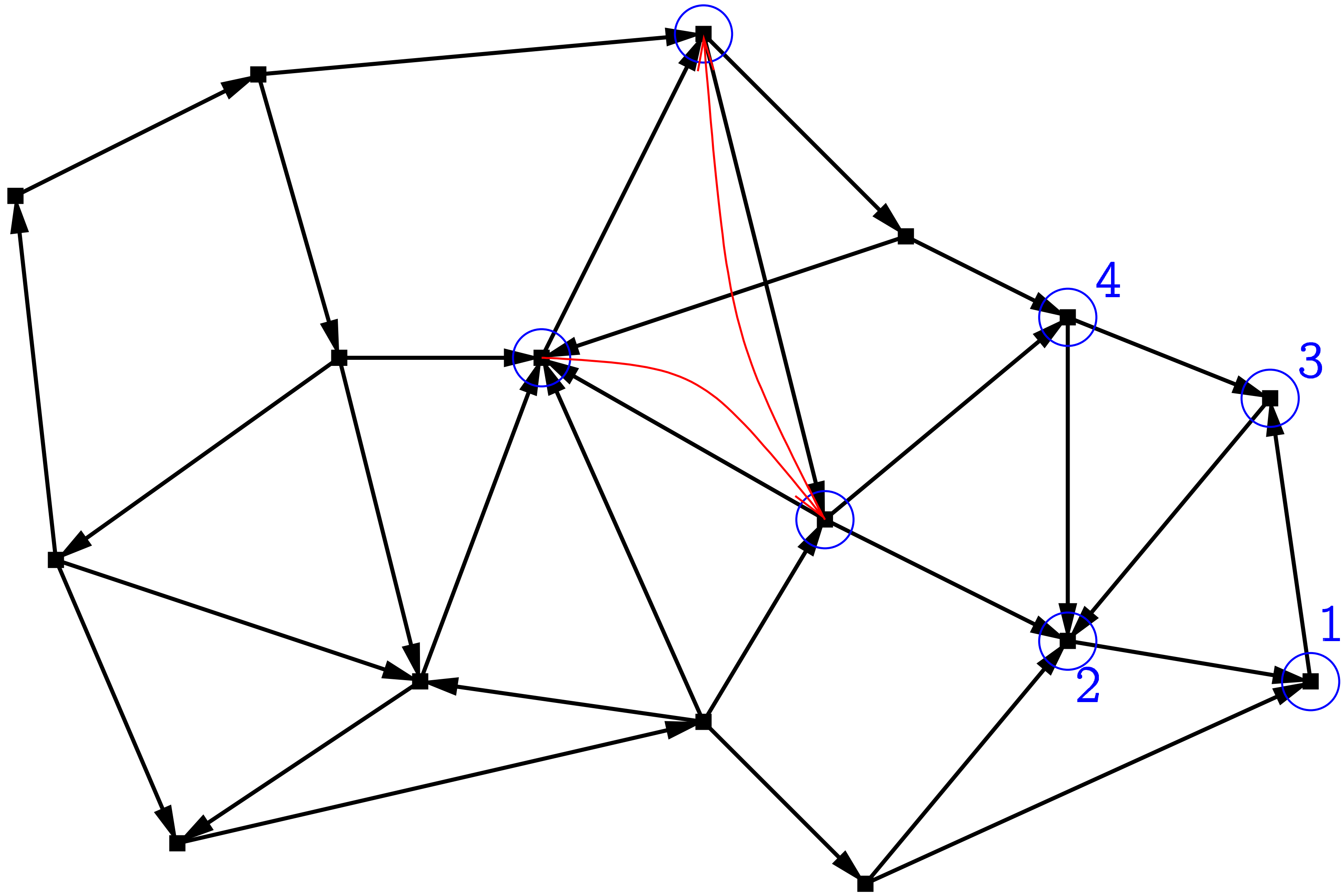



1

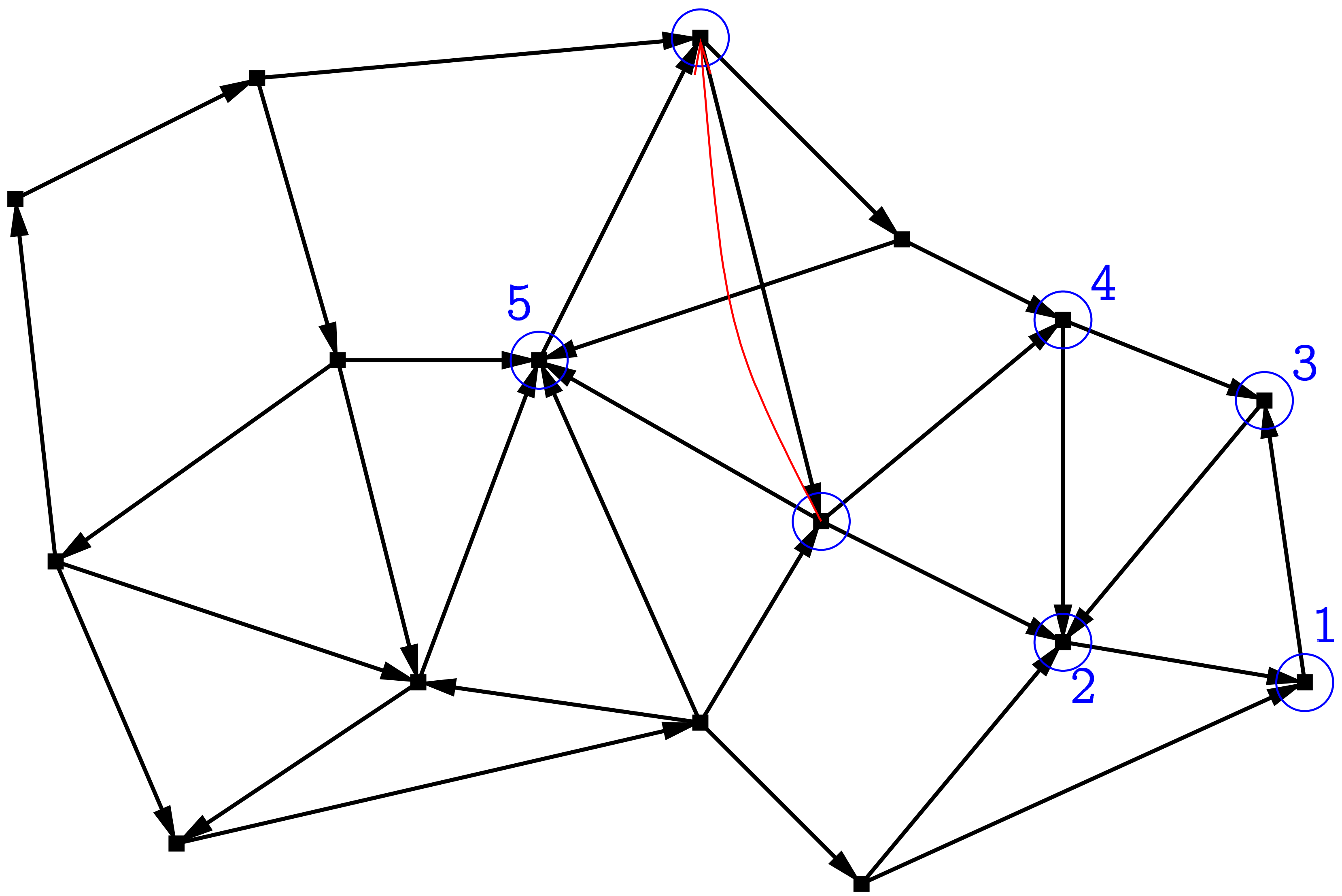

1
2


3
1
2

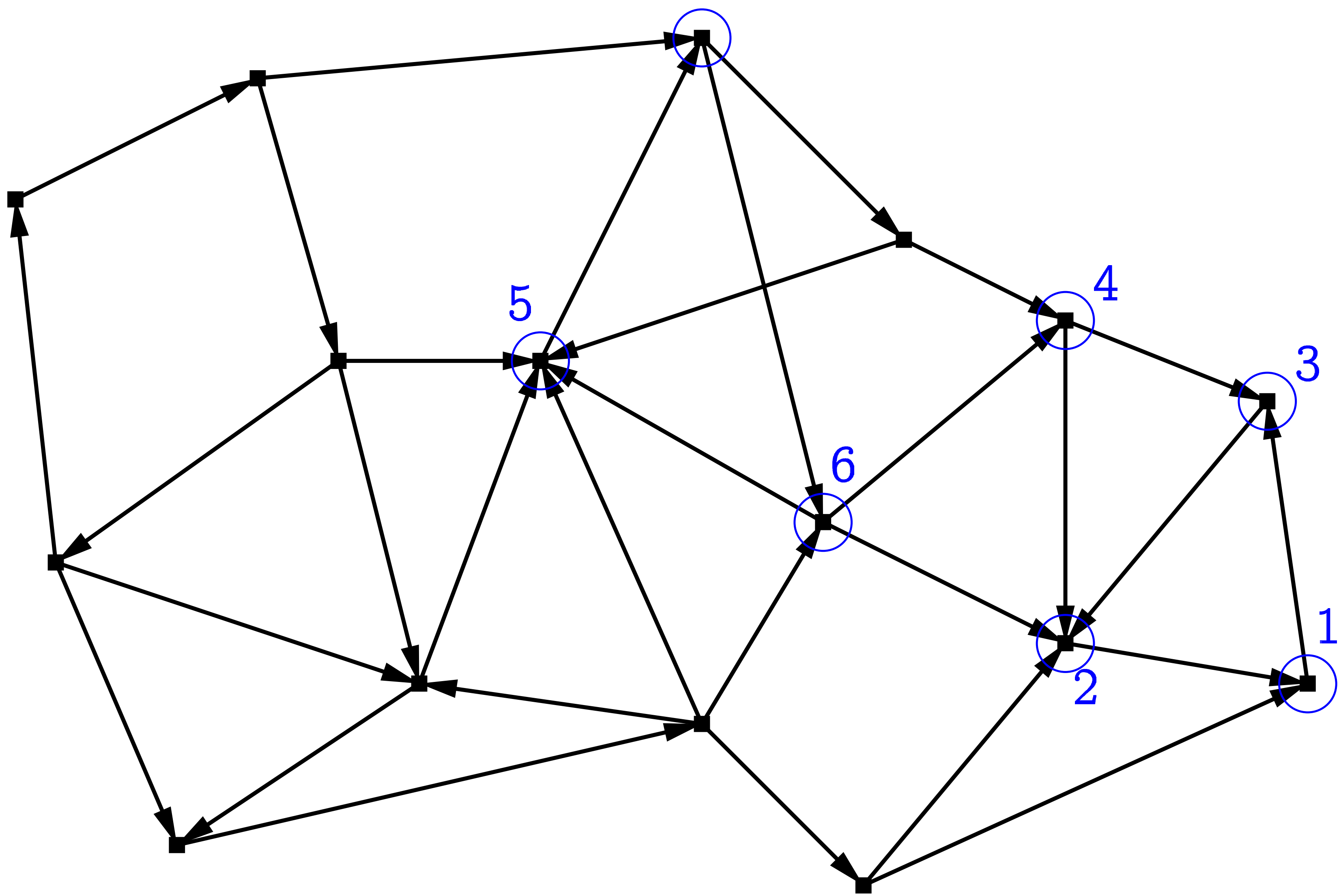

4
3
1
2

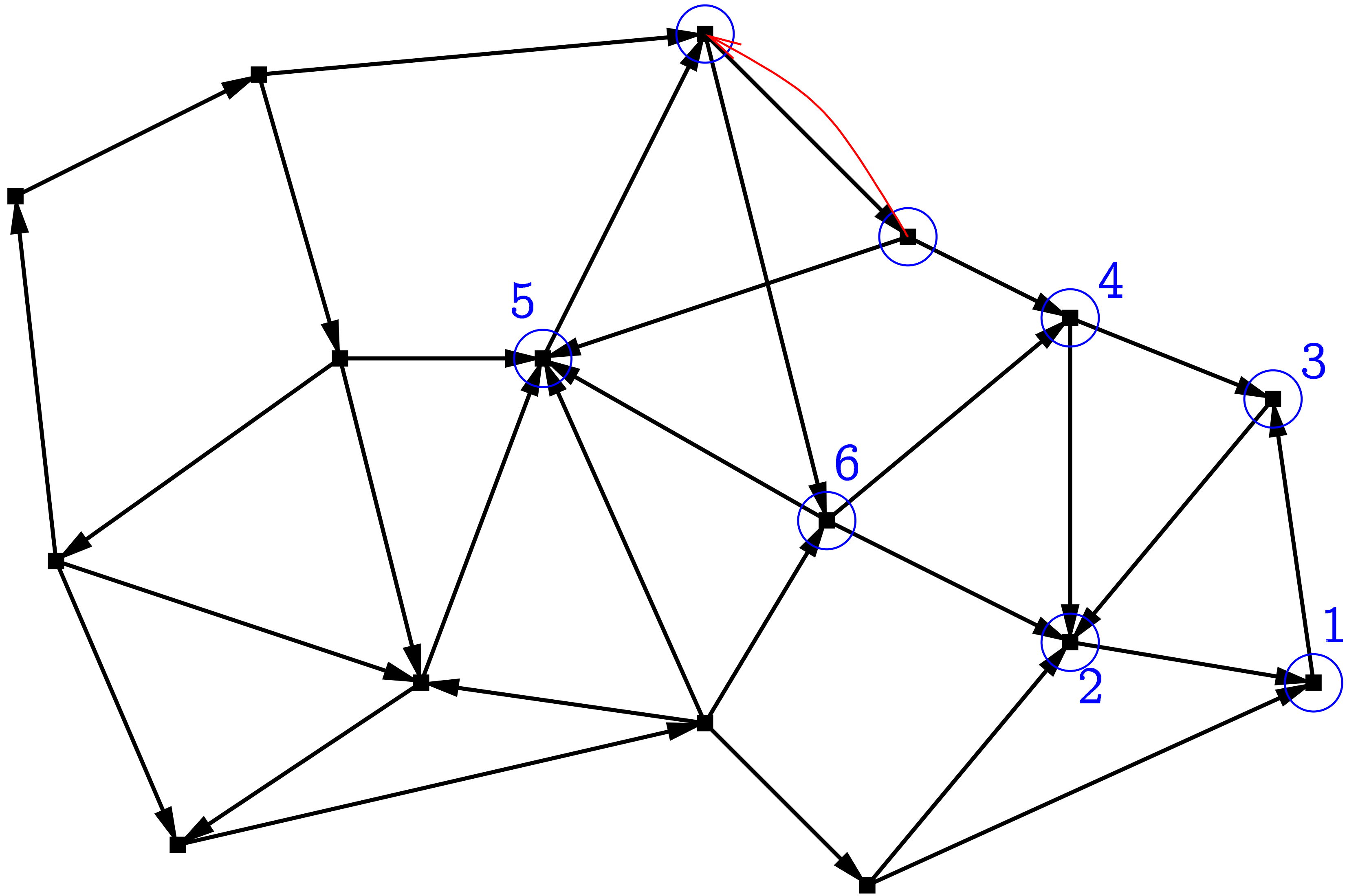


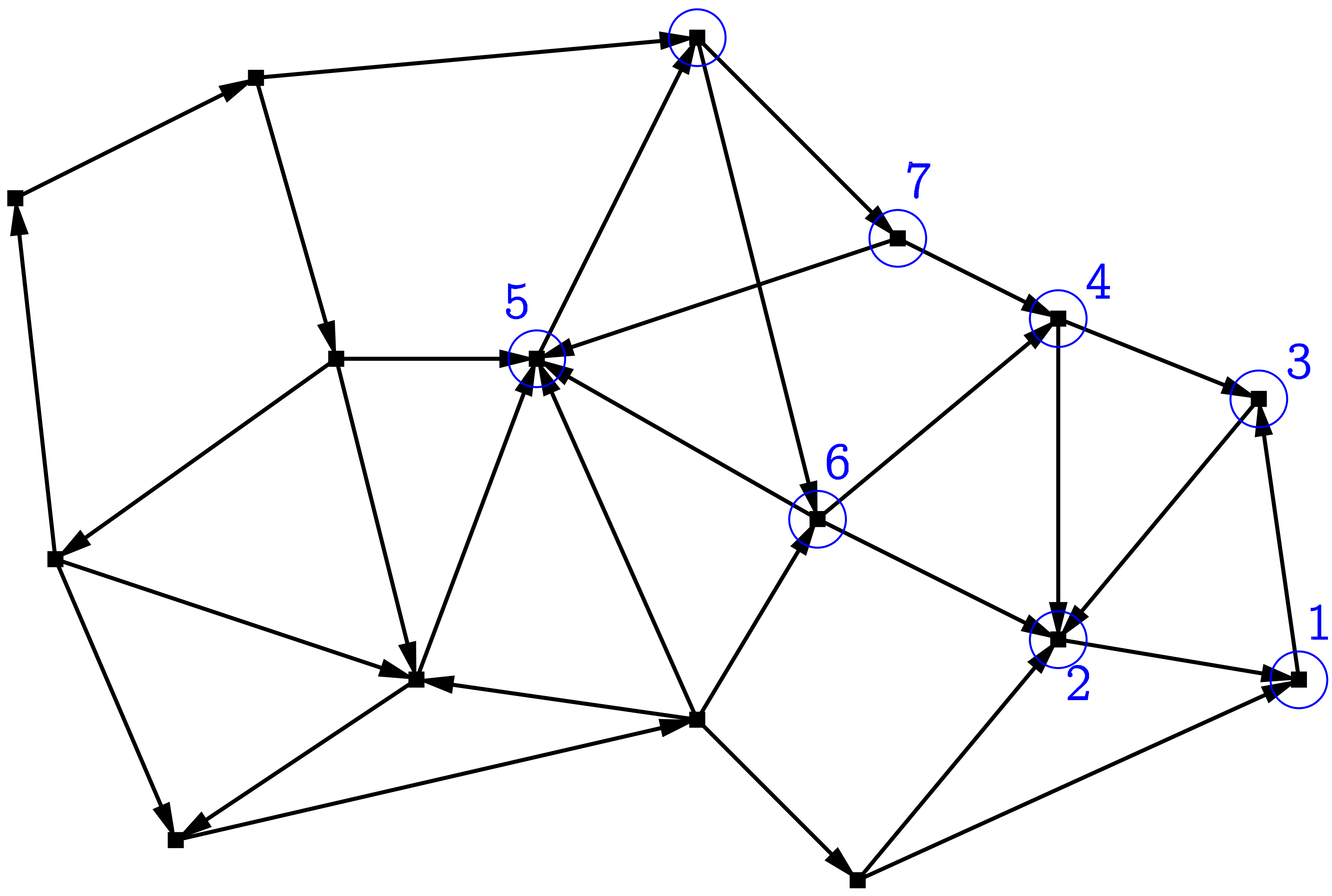
4
3
1
2

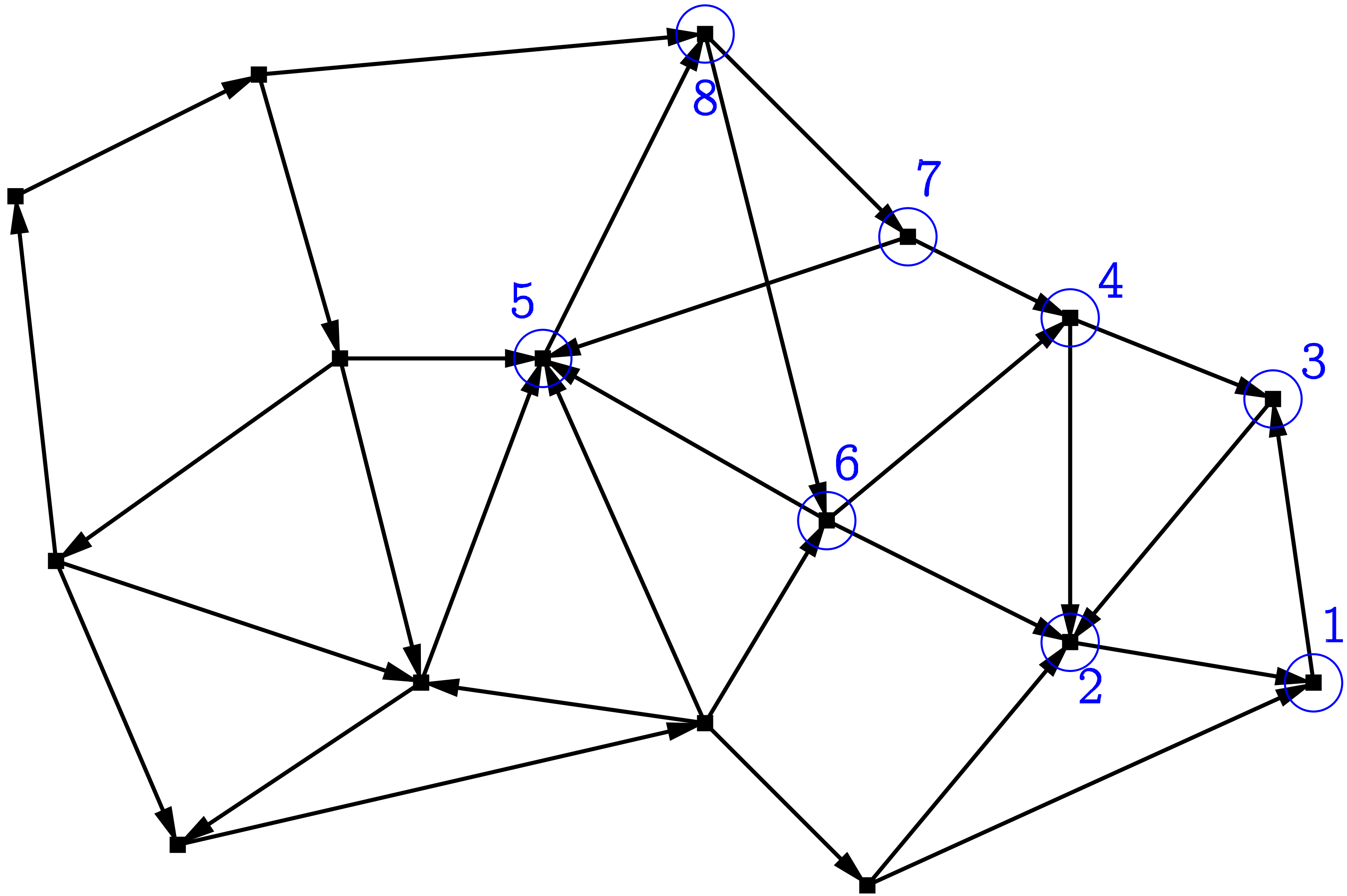


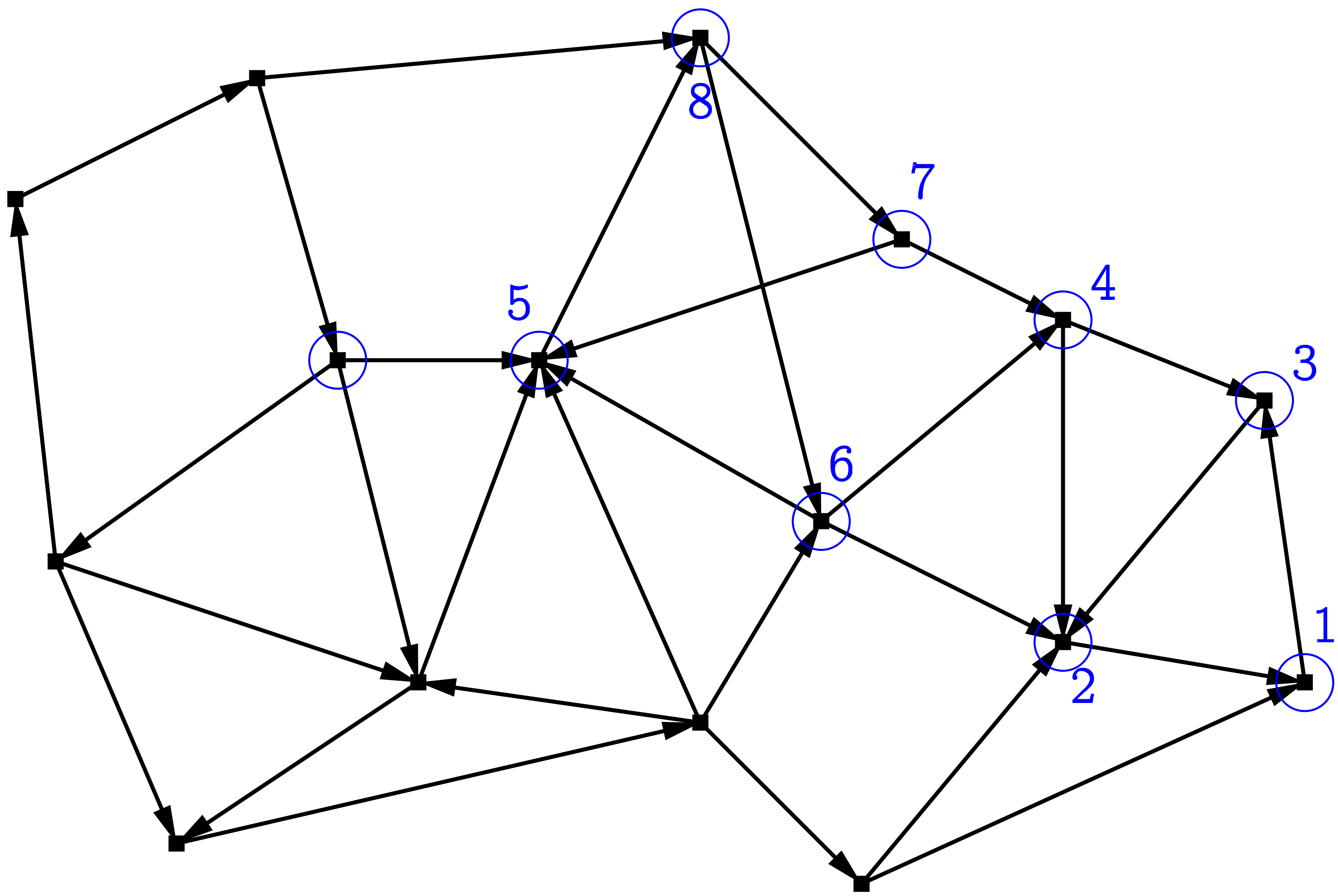
5
4
3
1
2

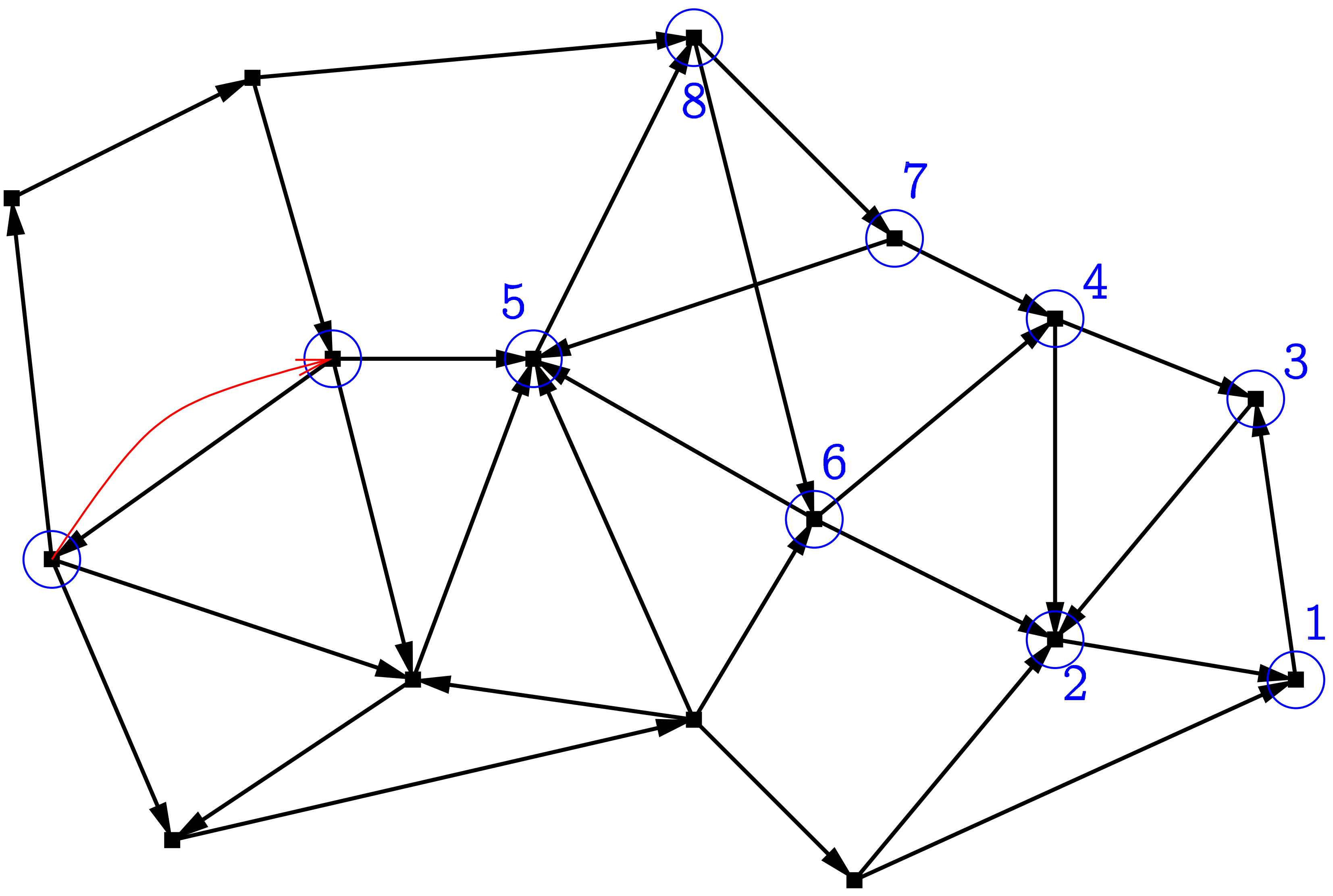

5
4
3
6
1
2

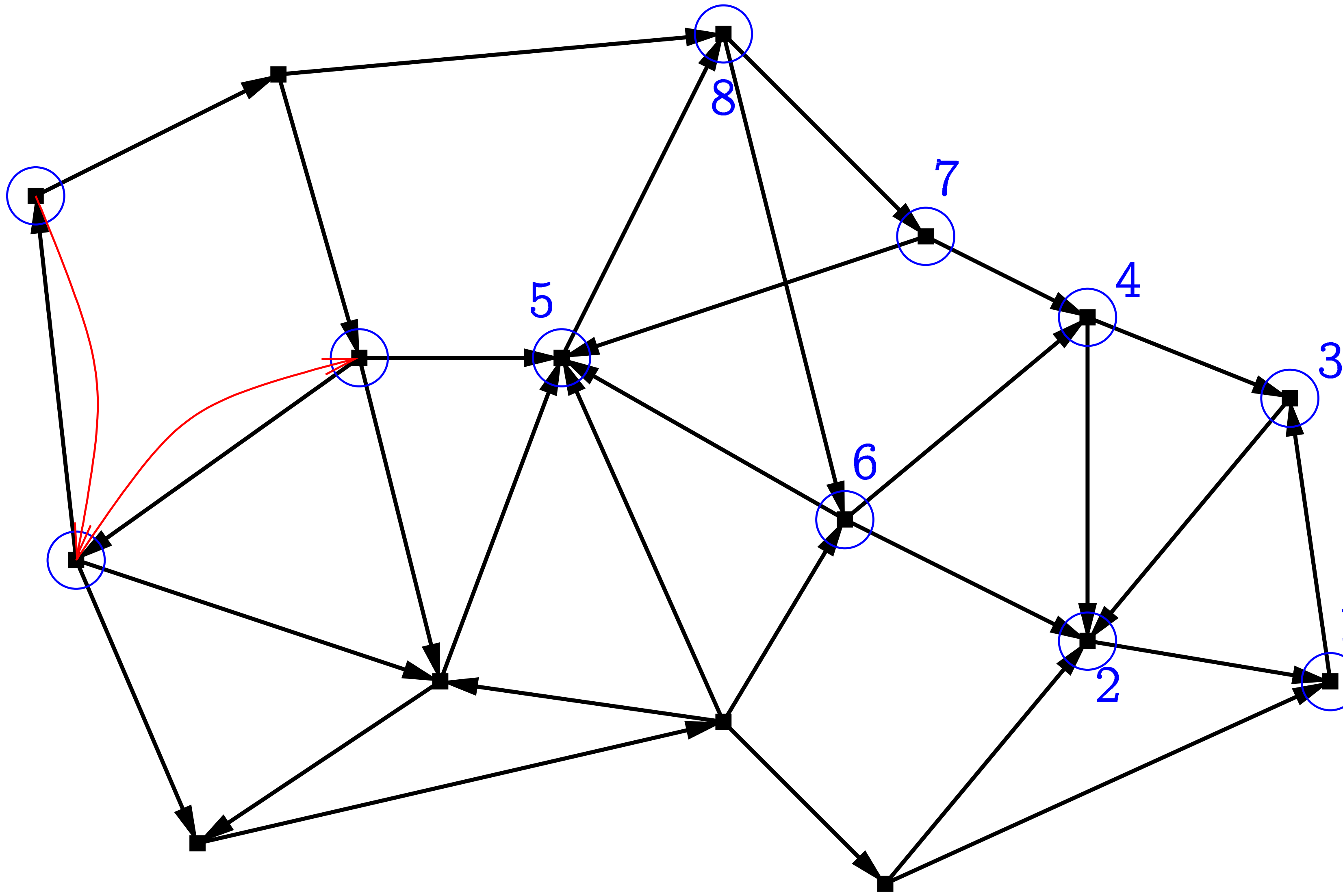

5
4
3
6
1
2

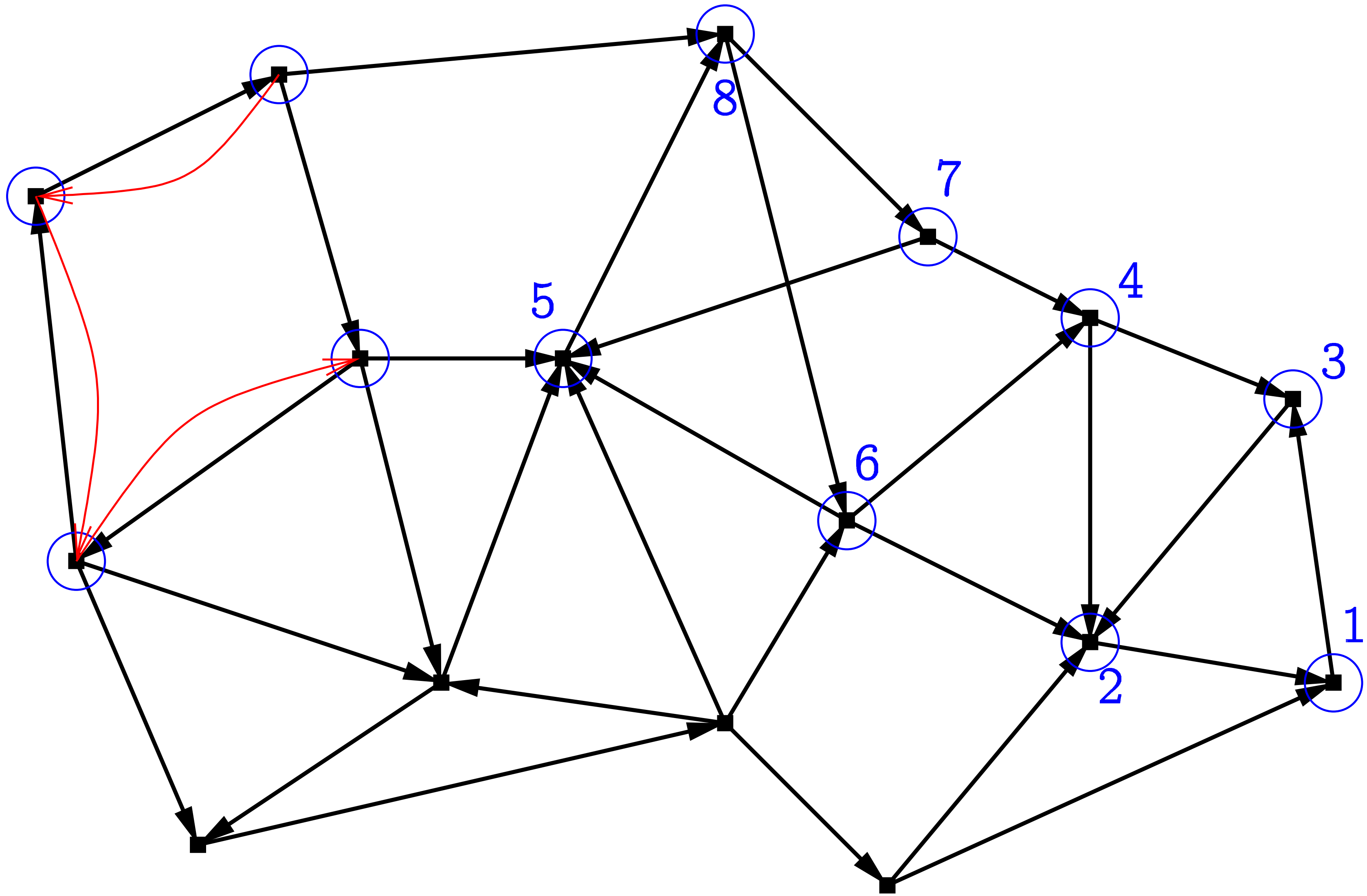

7
5
4
3
6
1
2

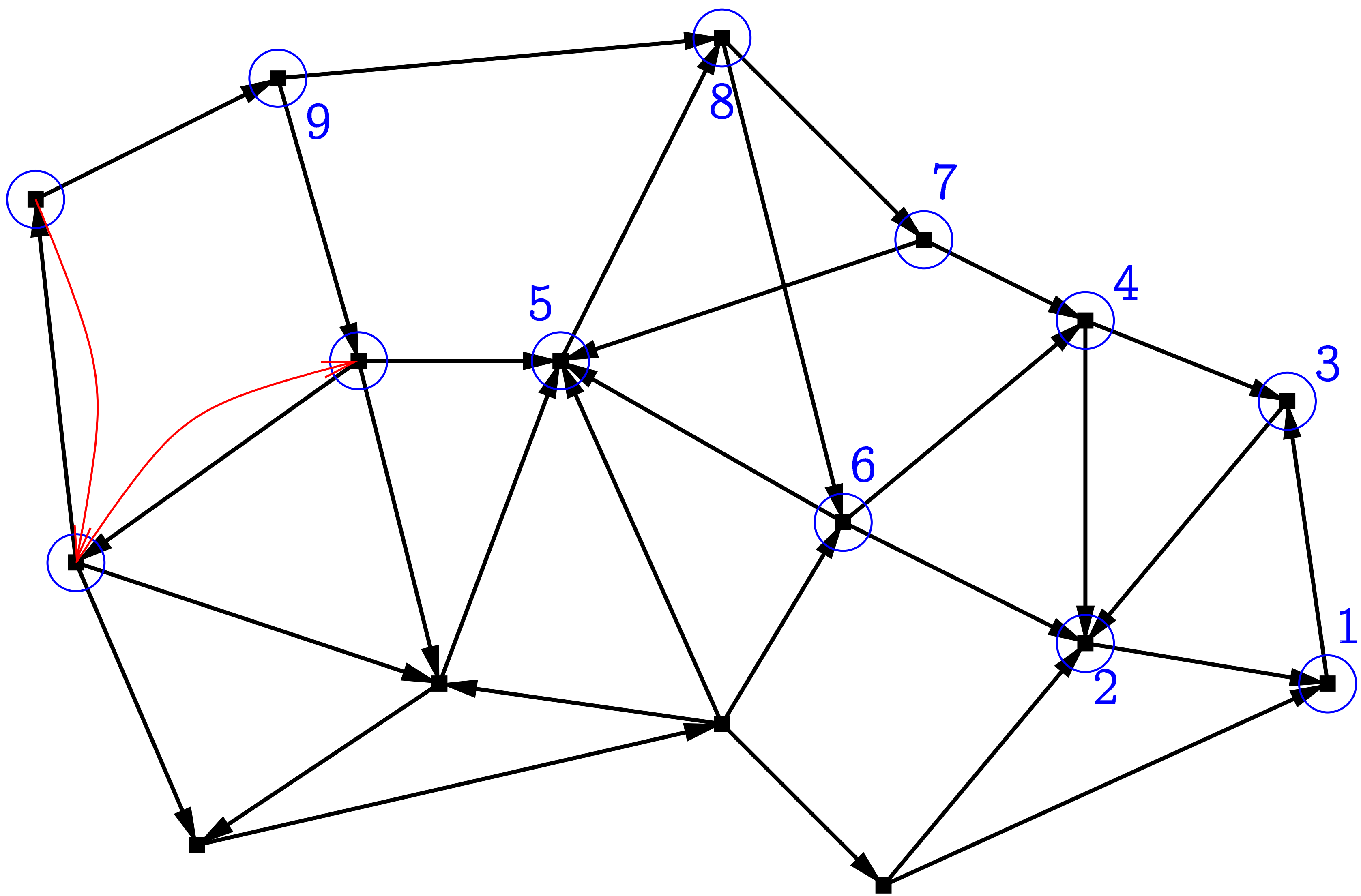

8
7
5
4
3
6
1
2


8
7
4
5
3
6
1
2

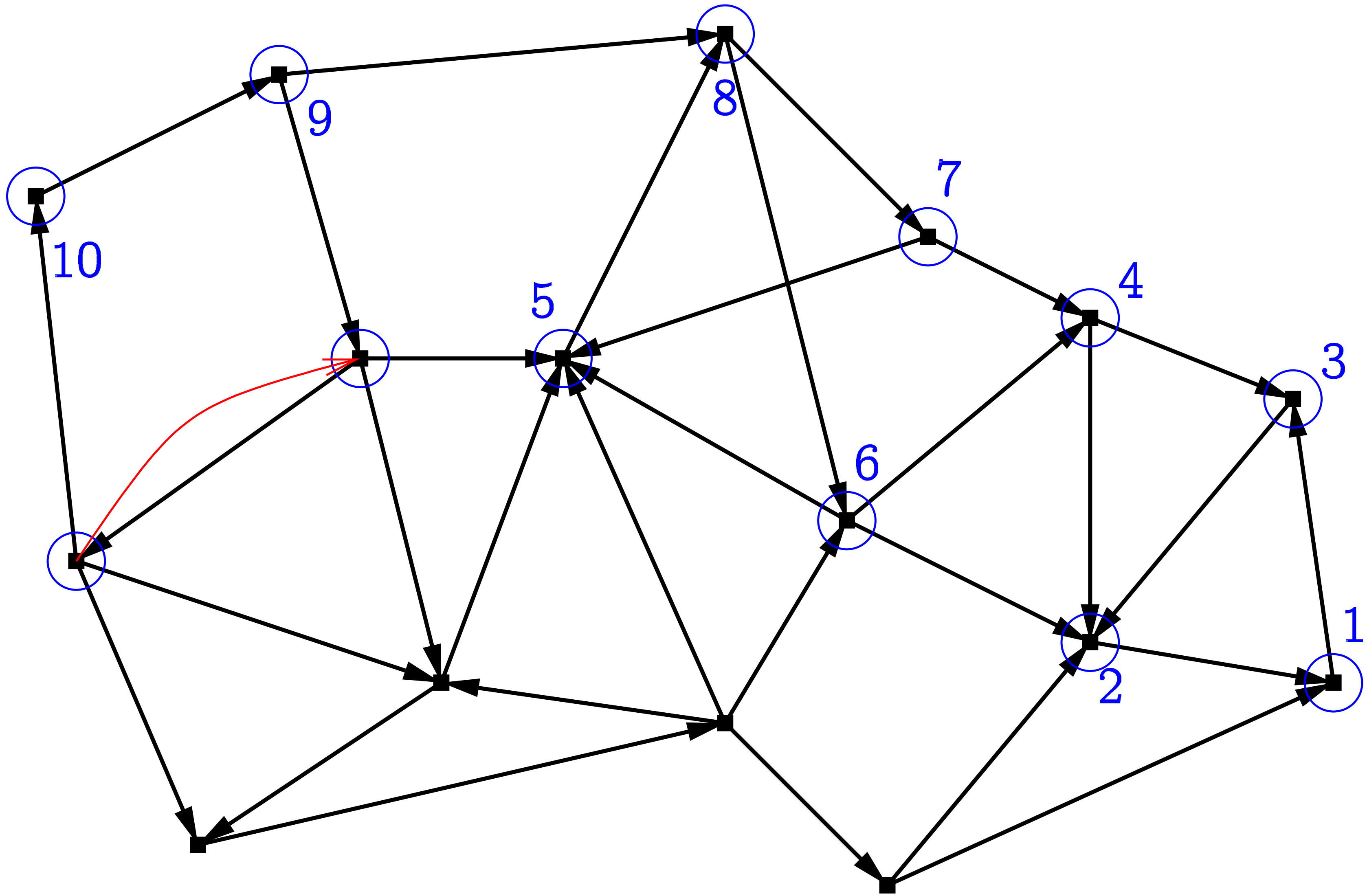

8
7
4
5
3
6
1
2


8
7
5
4
3
6
1
2

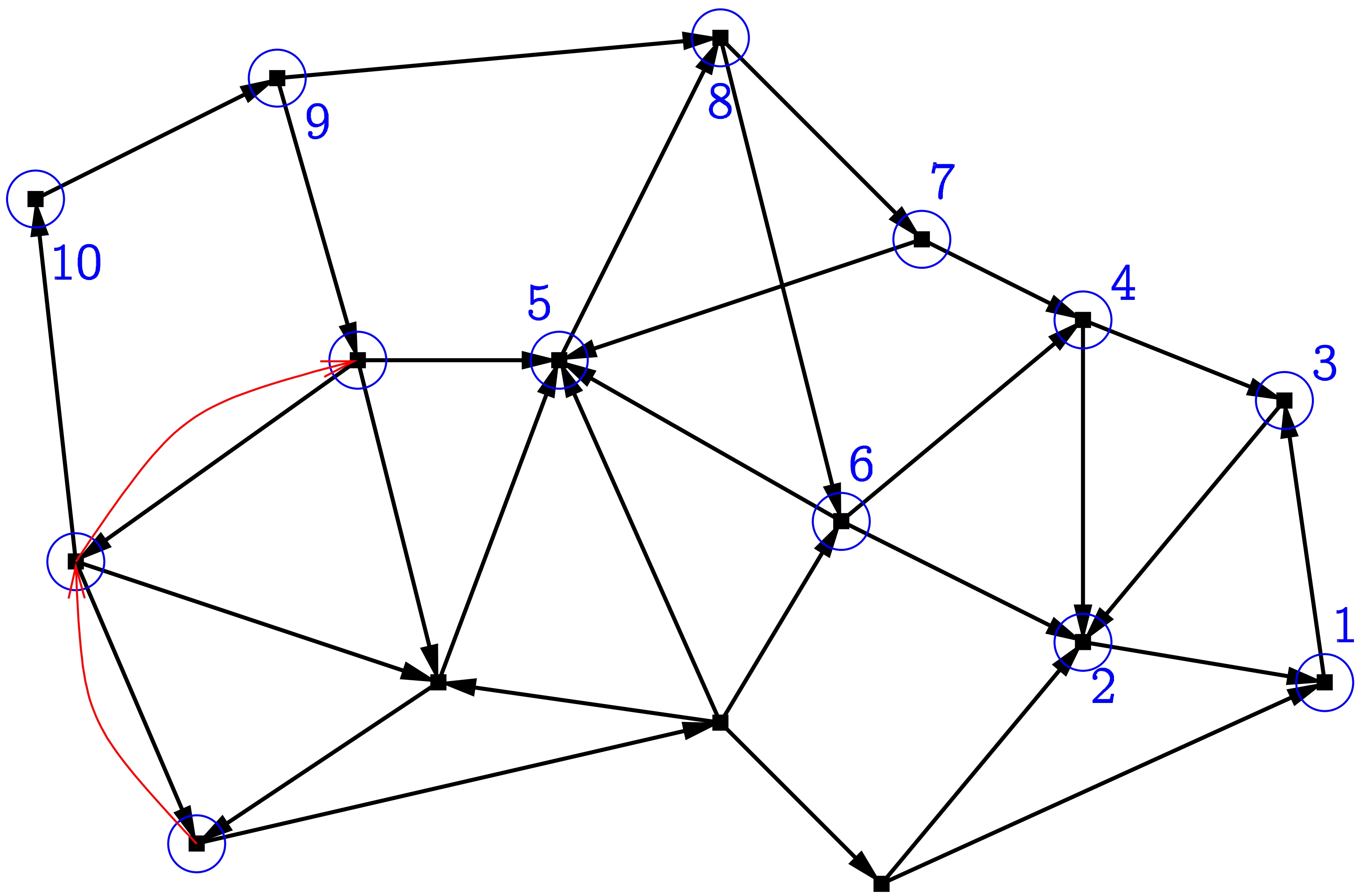

8
7
5
4
3
6
1
2



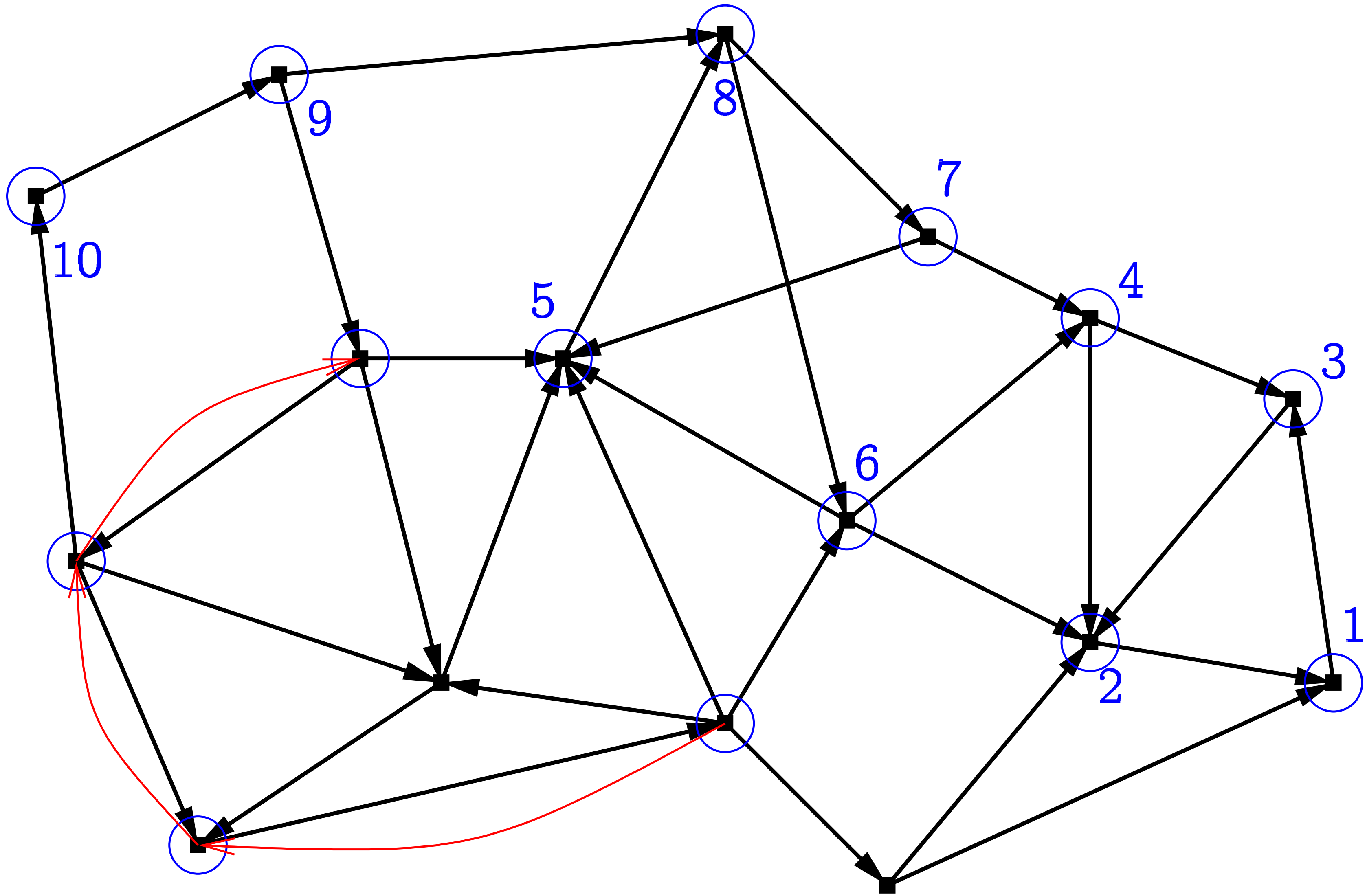
9
8
7
5
4
3
6
1
2



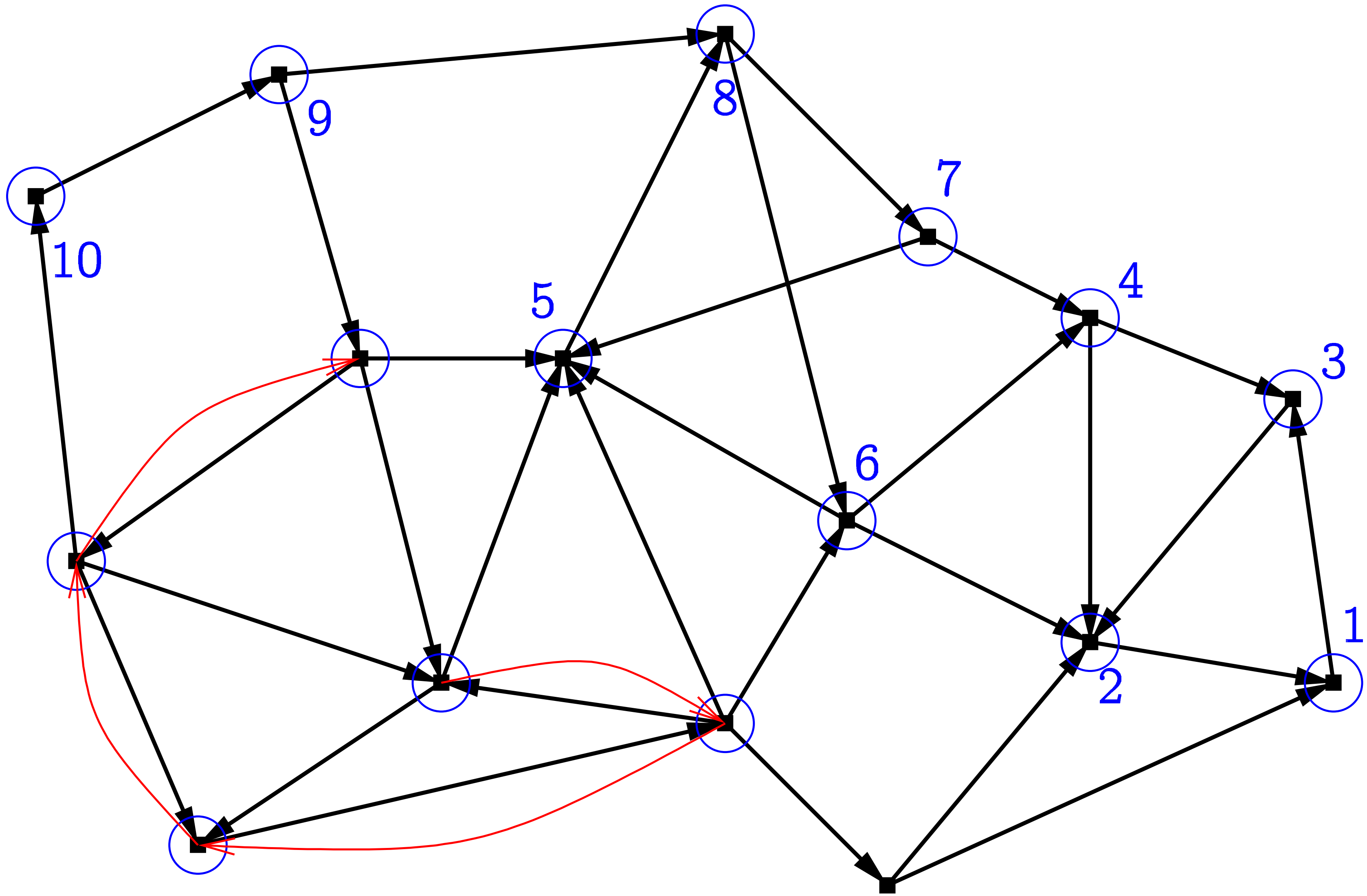
8
9
7
10
4
5
3
6
1
2



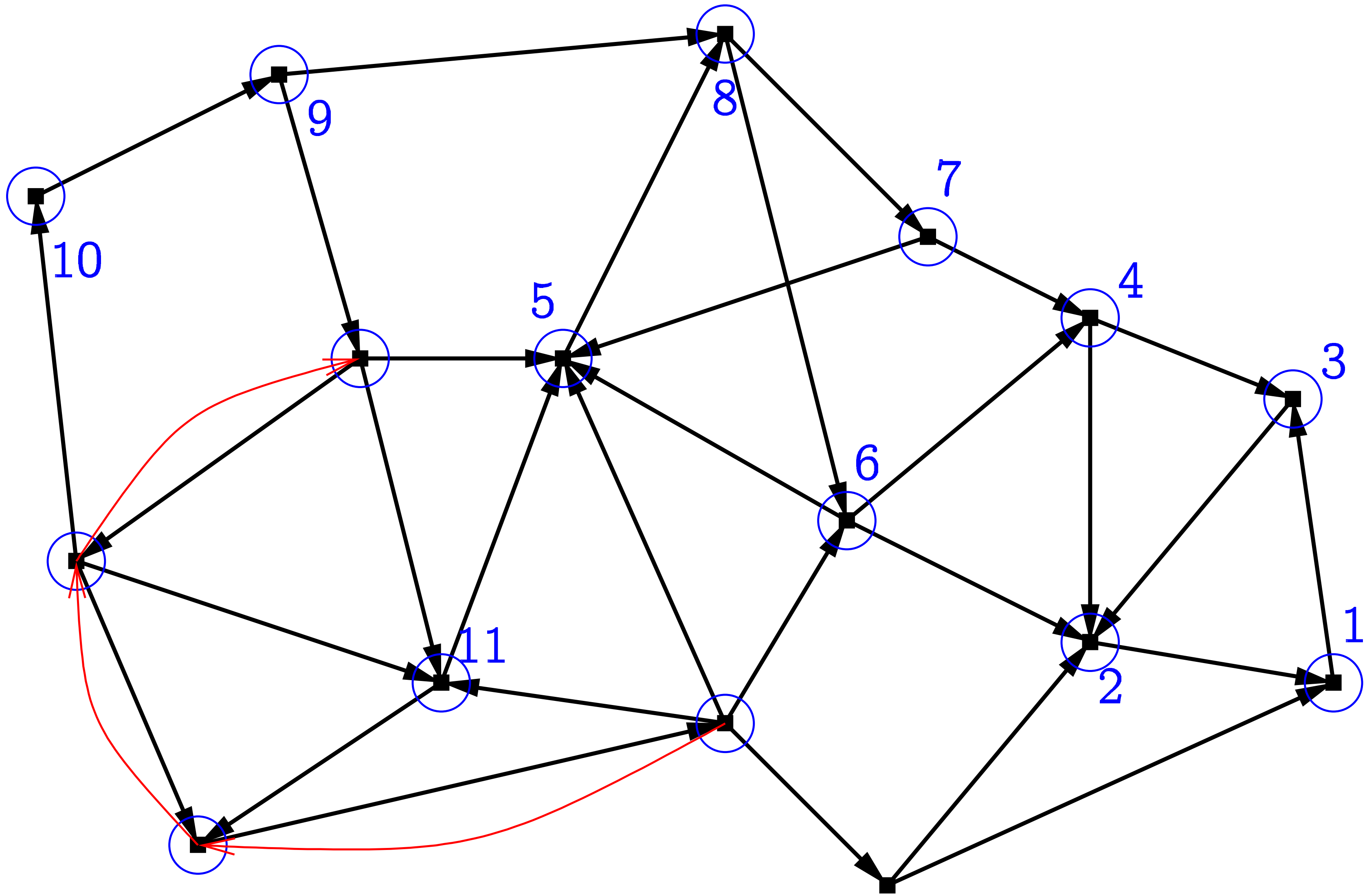
10
9
8
7
5
4
3
6
1
2



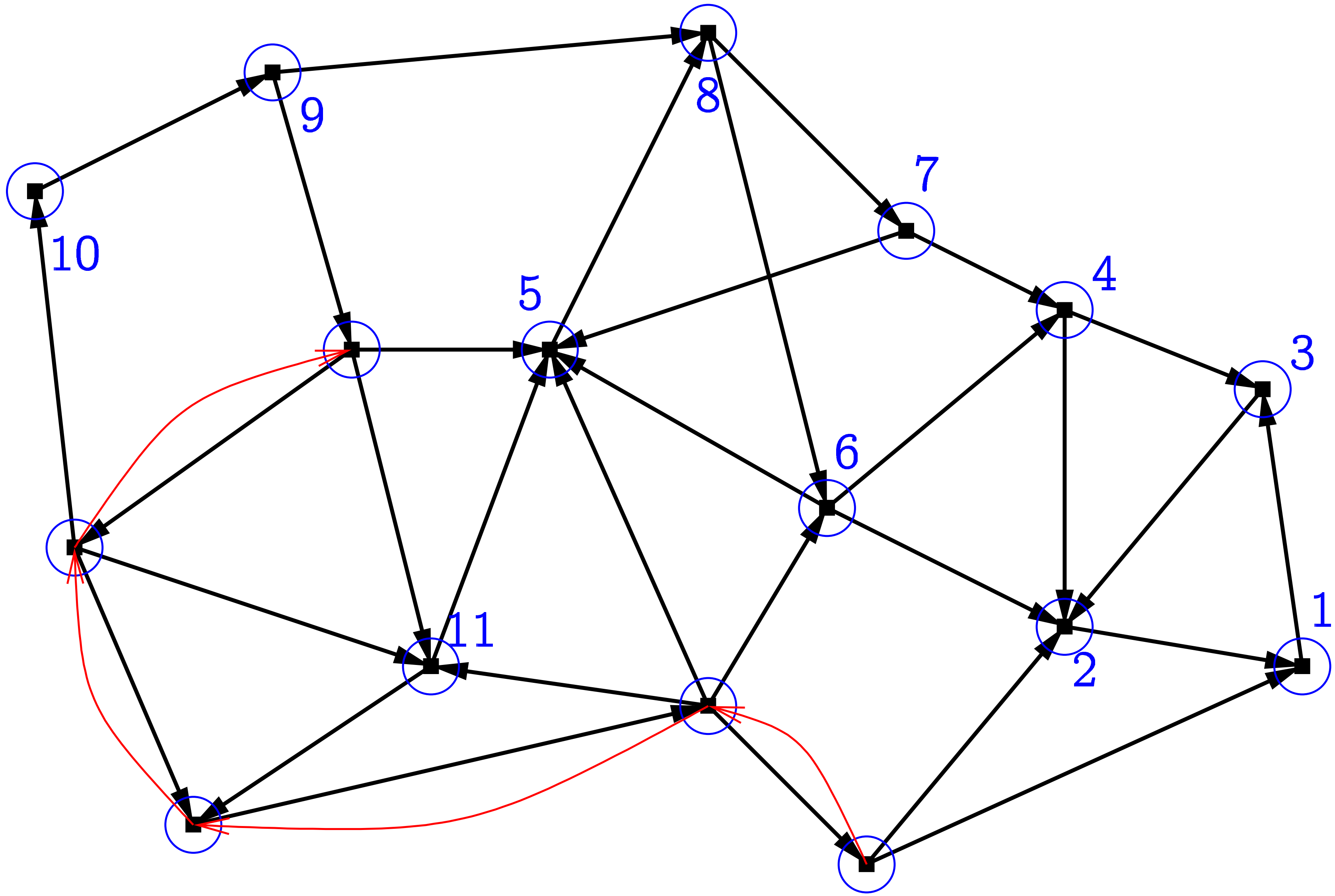
9
8
7
10
5
4
3
6
1
2



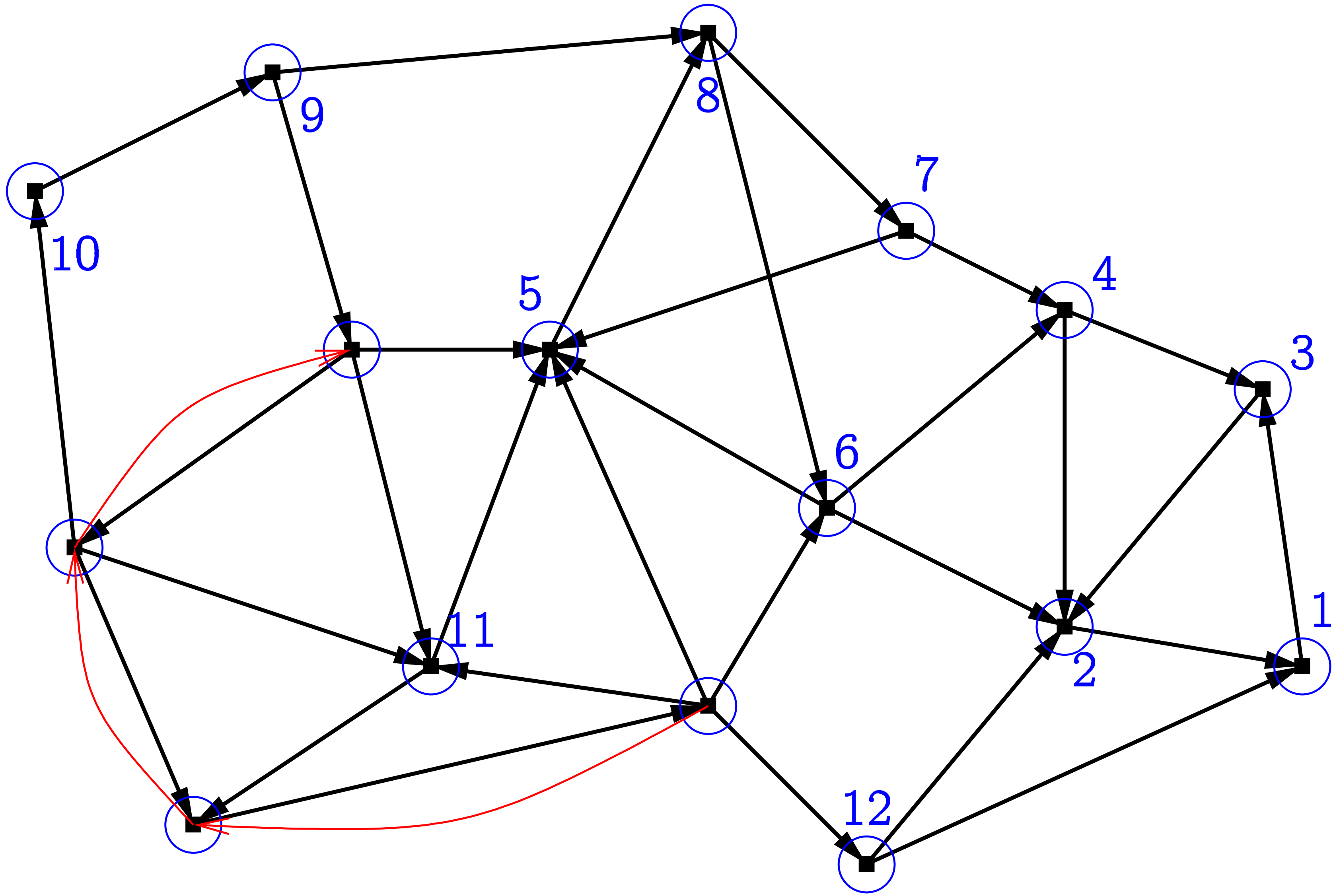
10
9
8
7
5
4
3
6
1
2



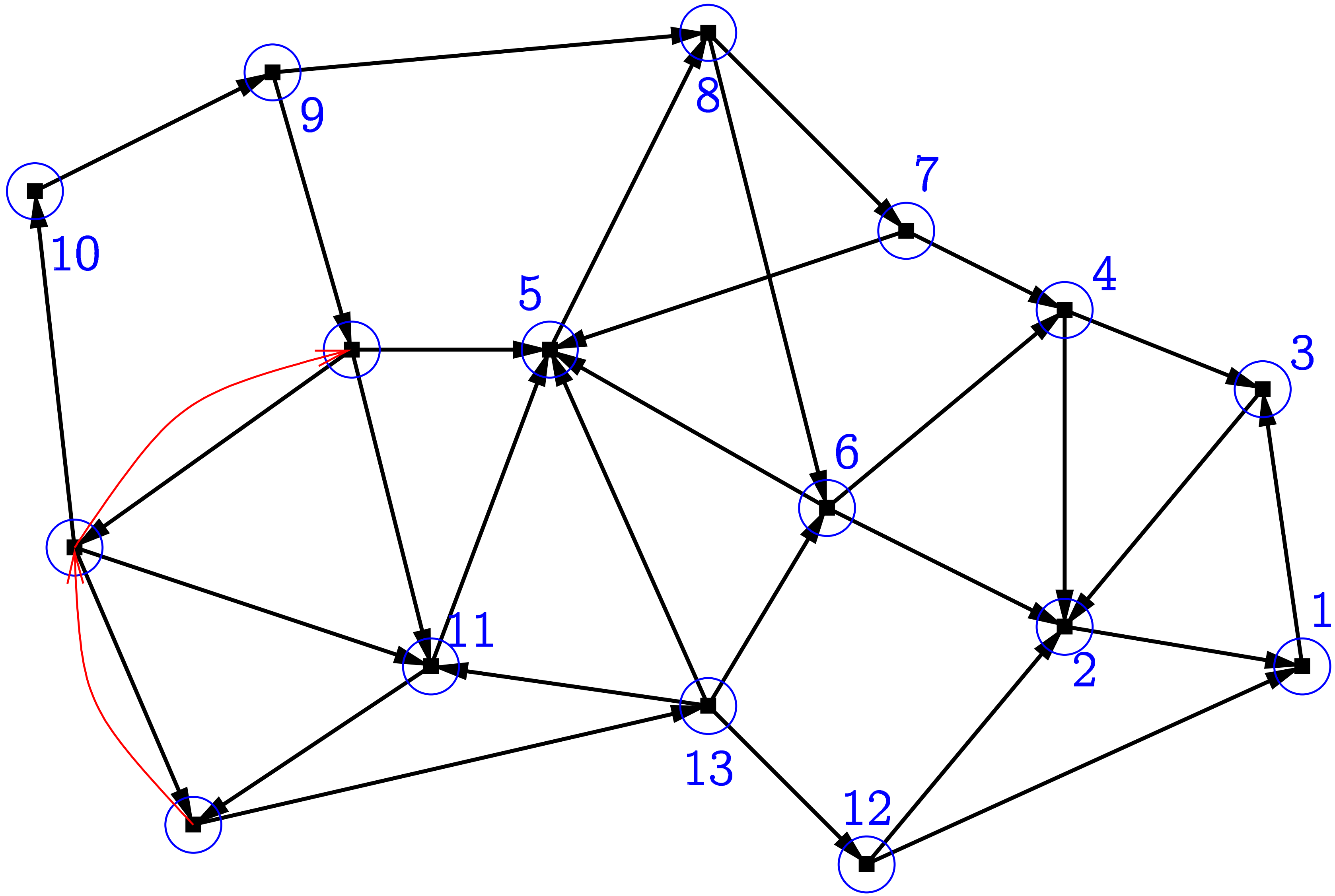
1
2
3
4
5
6
7
8
9
10
11



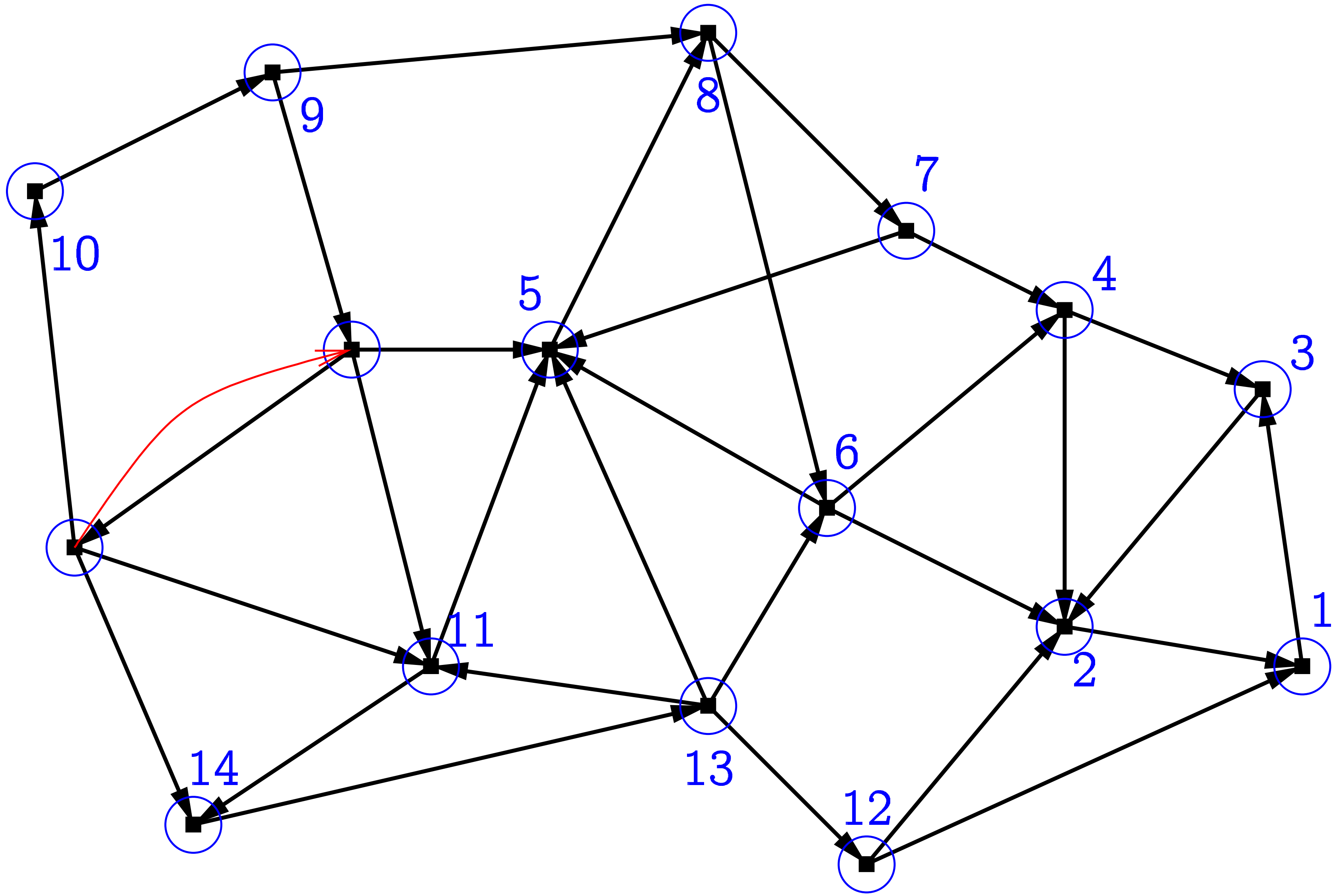
1
2
3
4
5
6
7
8
9
10
11



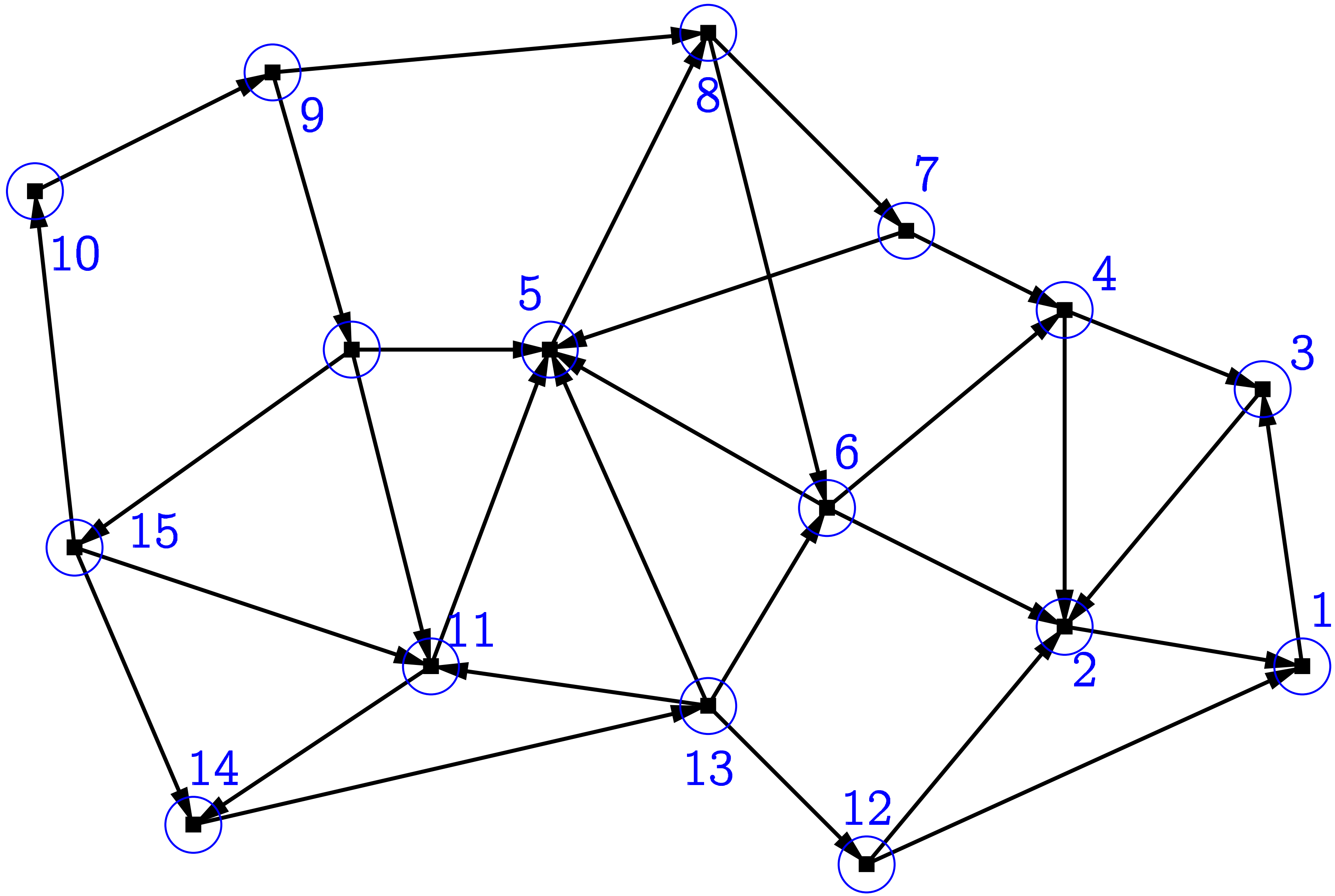
1
2
3
4
5
6
7
8
9
10
11
12



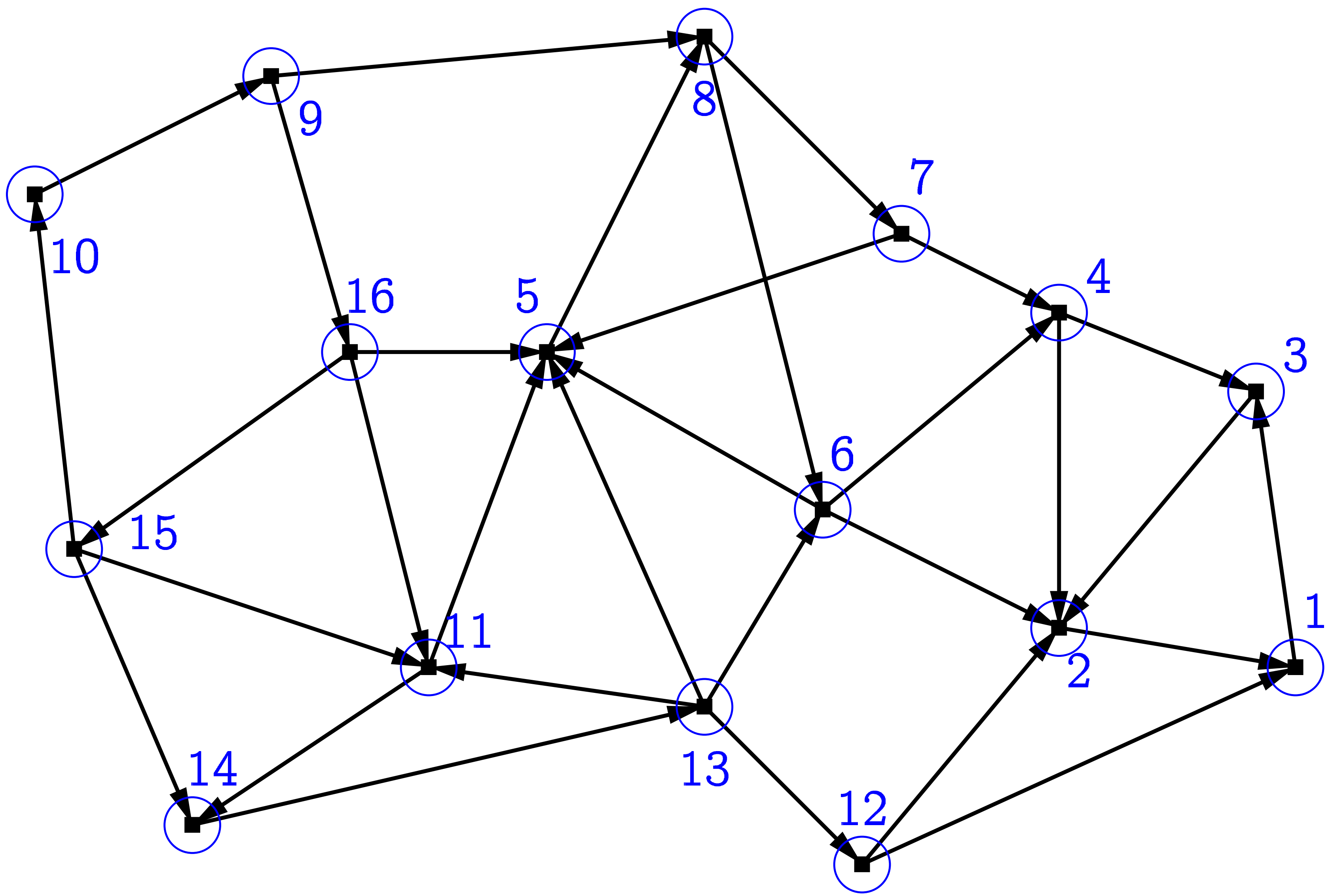
8
9
7
10
5
4
3
6
1
11
2
13
12



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



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



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

Tähelepanekuid algoritmi kohta:

Iga tipu v jaoks, mille jaoks „sügavuti“ 4. real „külasta“ välja kutsutakse, käiakse enne tagasipöördumist läbi kõik tipud, kuhu v-st jõuda võib ja mis ei ole veel läbi käidud.

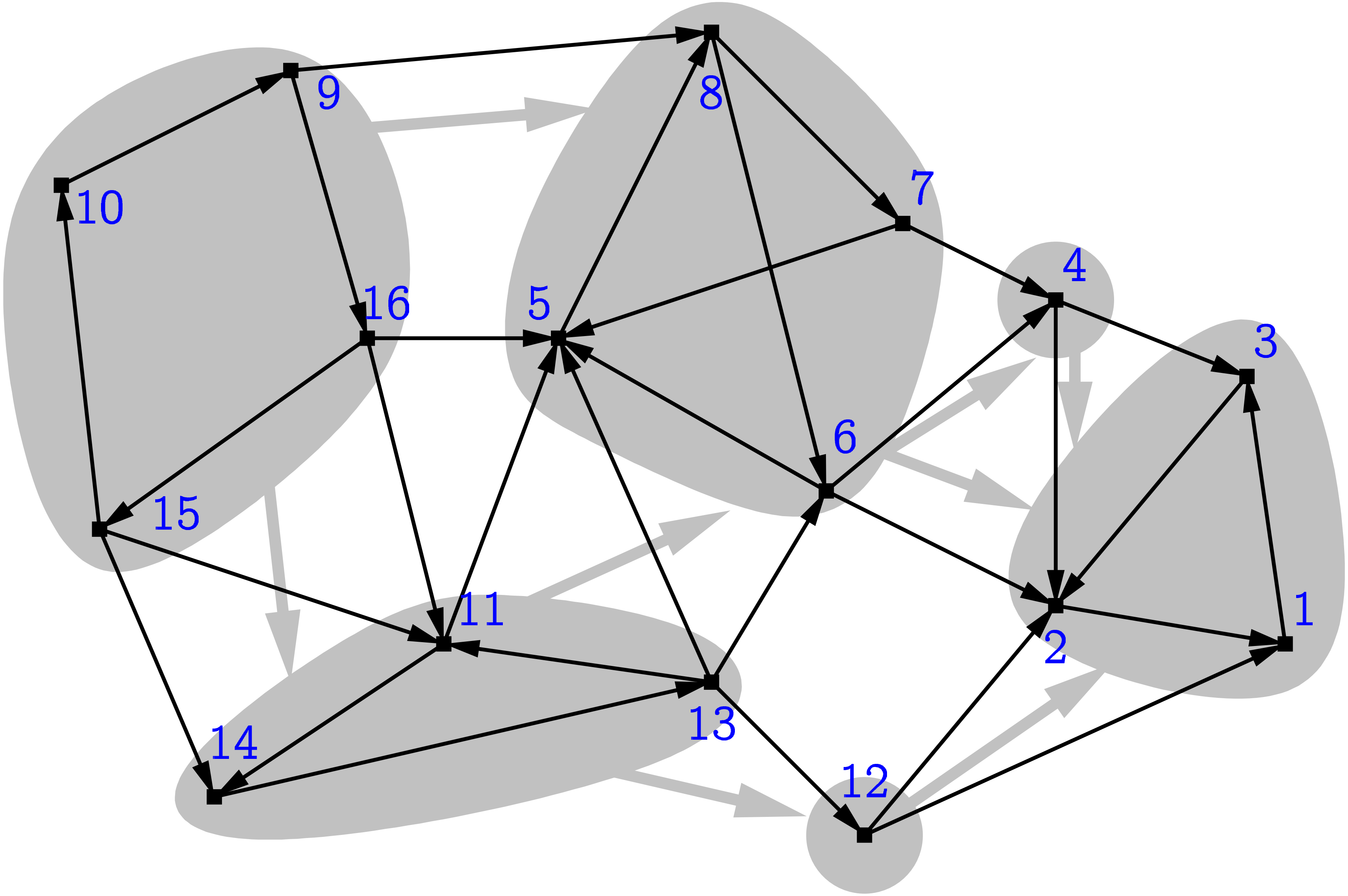
S.t. iga v jaoks, mille jaoks „külasta“ kutsuti välja „sügavuti“-st, ja iga w jaoks, kuhu on v-st võimalik minna, kehtib $v.jrknr > w.jrknr$.

Iga tugevalt sidus komponent käiakse läbi sama „külasta“ „sügavuti“-st väljakutse ajal.

Kui v on esimene tipp mingist tugevalt sidusast komponendist, mille jaoks „külasta“ välja kutsutakse, siis enne tagasipöördumist käiakse läbi see tugevalt sidus komponent ning samuti kõik tugevalt sidusad komponendid, kuhu sealt saab ning mis ei ole veel läbi käidud.

Olgu V_1, V_2 kaks tugevalt sidusat komponenti, nii et V_1-st saab V_2-e. Olgu $v_1 \in V_1$ ja $v_2 \in V_2$ max. $.jrknr$-ga tipud neist komponentides. Siis $v_1.jrknr > v_2.jrknr$.

Kui seame G^{komp} igale tipule (s.t. esialgse graafi tugevalt sidusale komponendile) $U \subseteq V$ vastavusse arvu $\max\limits_{v \in U} v.jrknr$, siis saame G^{komp} tipud topoloogiliselt sorteeritud (kahanevas järjekorras).

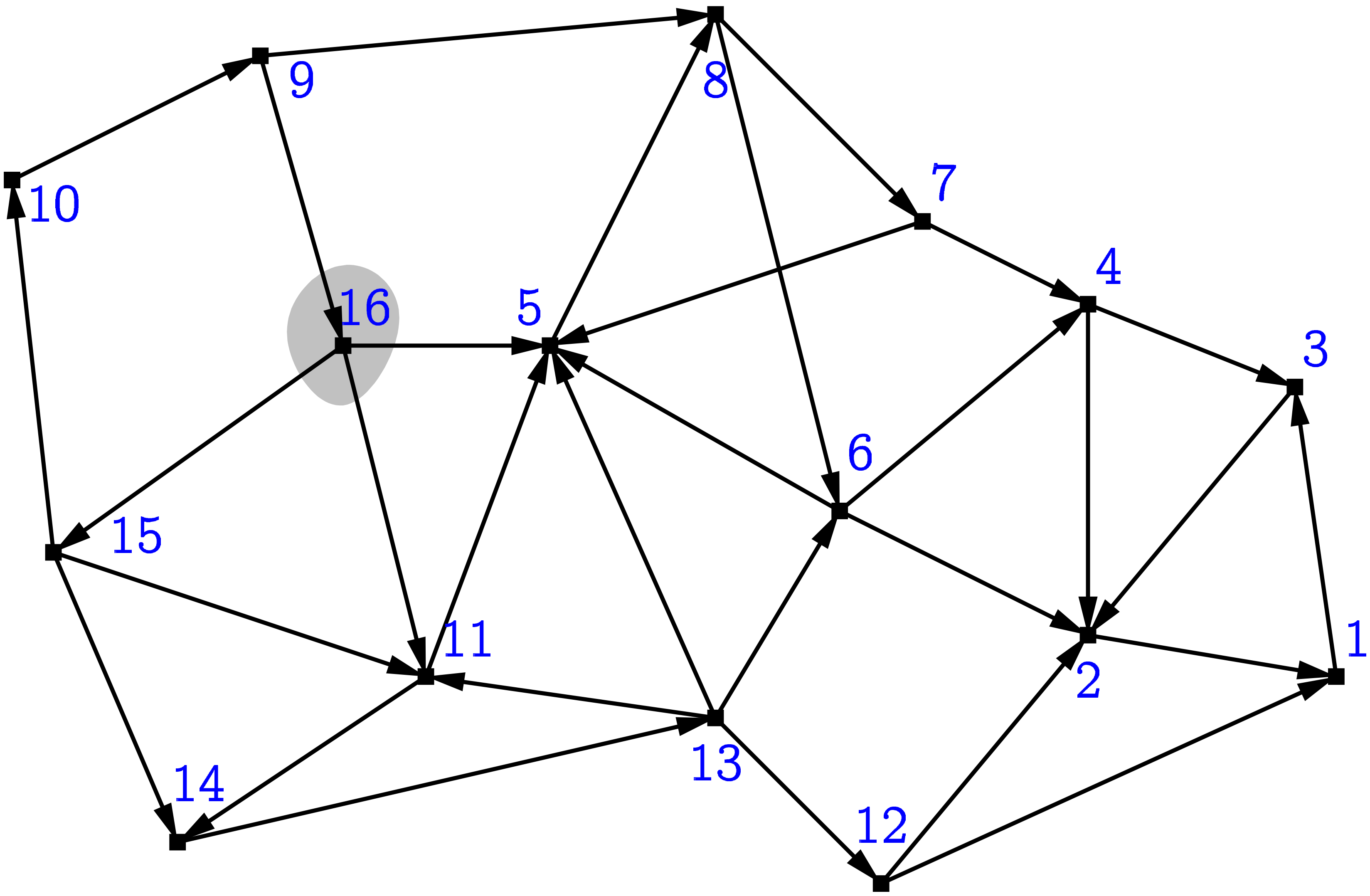

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

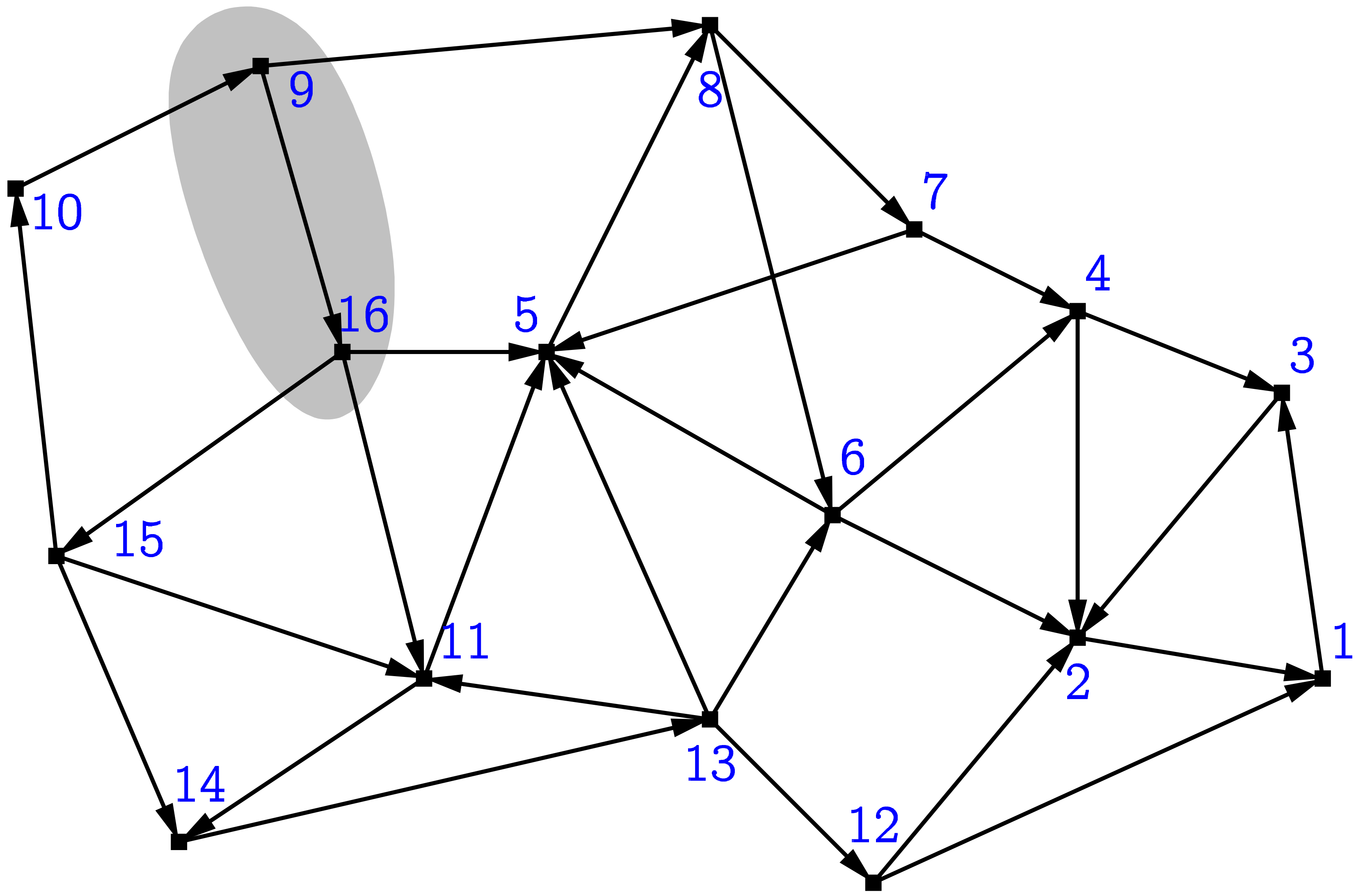
Algoritm tugevalt sidusate komponentide leidmiseks:

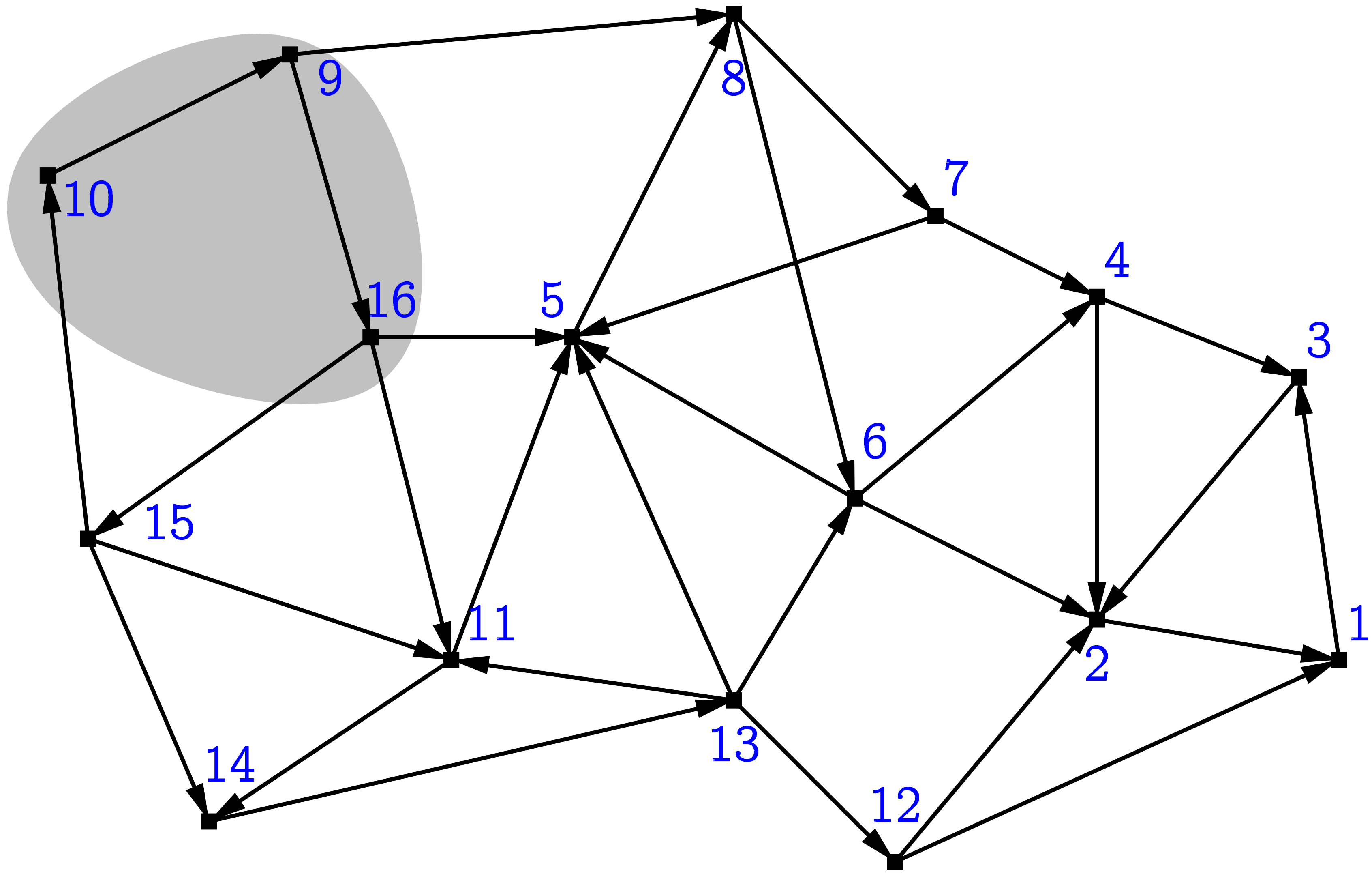
sügavuti(G)
$\textbf{for all } v \in V \textbf{ do } v.\mathit{läbitud} := \mathsf{false};\ s[v.\mathit{jrknr}] := v$
$K := \emptyset$
$\textbf{for } i := |V| \textbf{ downto } 1 \textbf{ do}$
$\quad \textbf{if } \neg s[i].\mathit{läbitud} \textbf{ then } K := K \cup \{\text{komponent}(s[i])\}$
$\textbf{return } K$

komponent(v) on

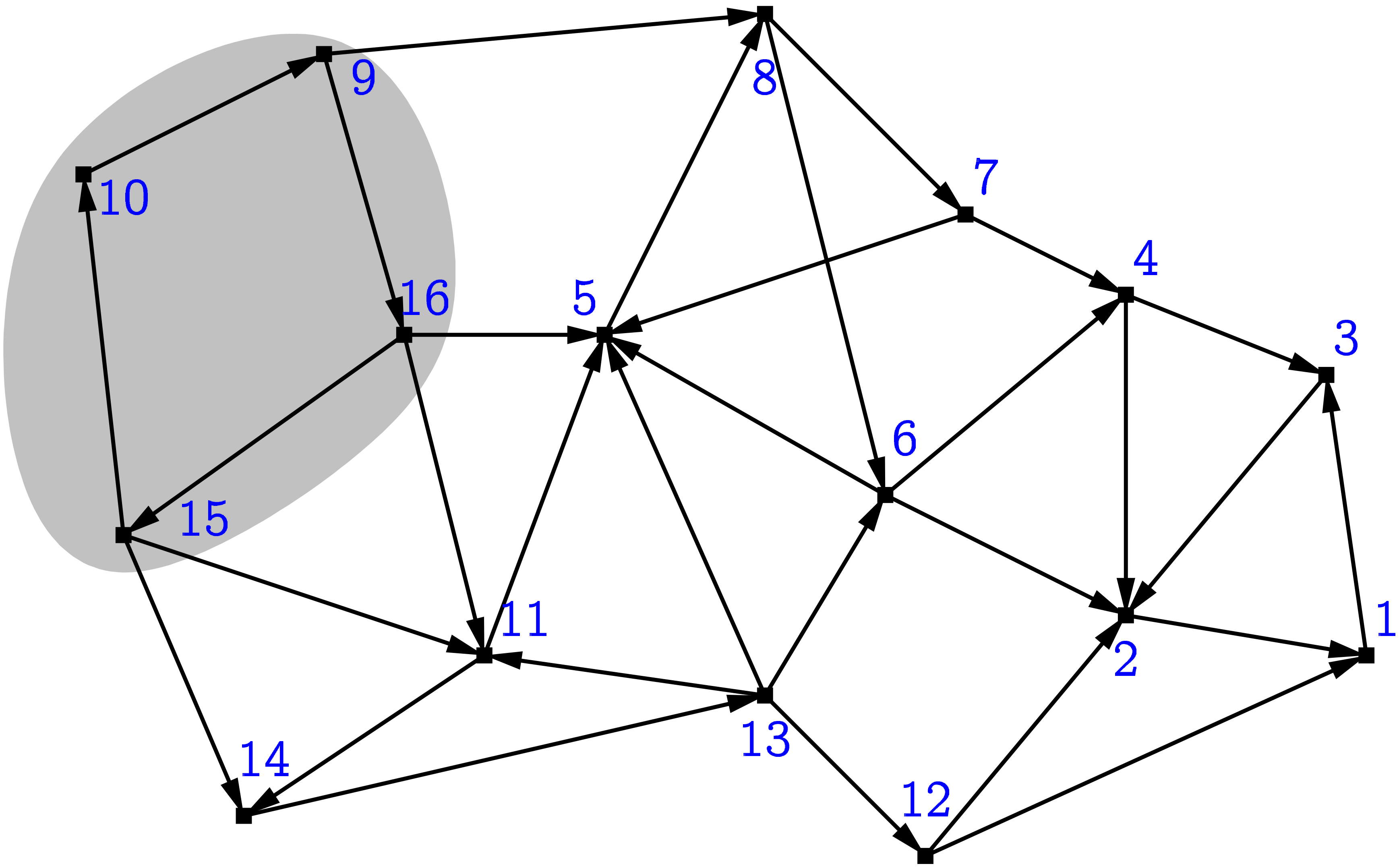
$v.\mathit{läbitud} := \mathsf{true}$
$k := \{v\}$
$\textbf{for all } w \in G^{-1}v \textbf{ do}$
$\quad \textbf{if } \neg w.\mathit{läbitud} \textbf{ then } k := k \cup \text{komponent}(w)$


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

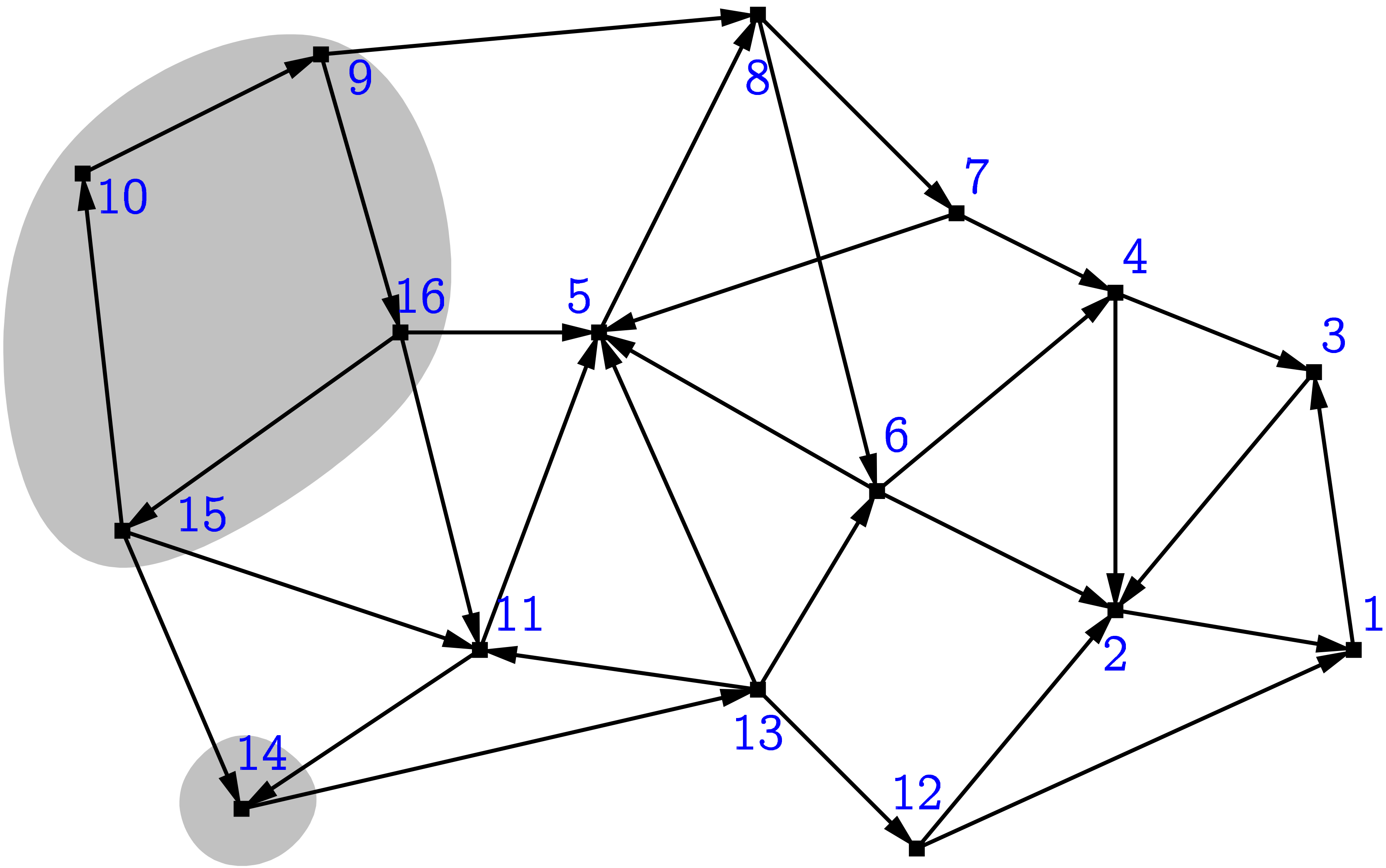

9
8
10
7
4
16
5
3
6
15
11
1
2
13
14
12



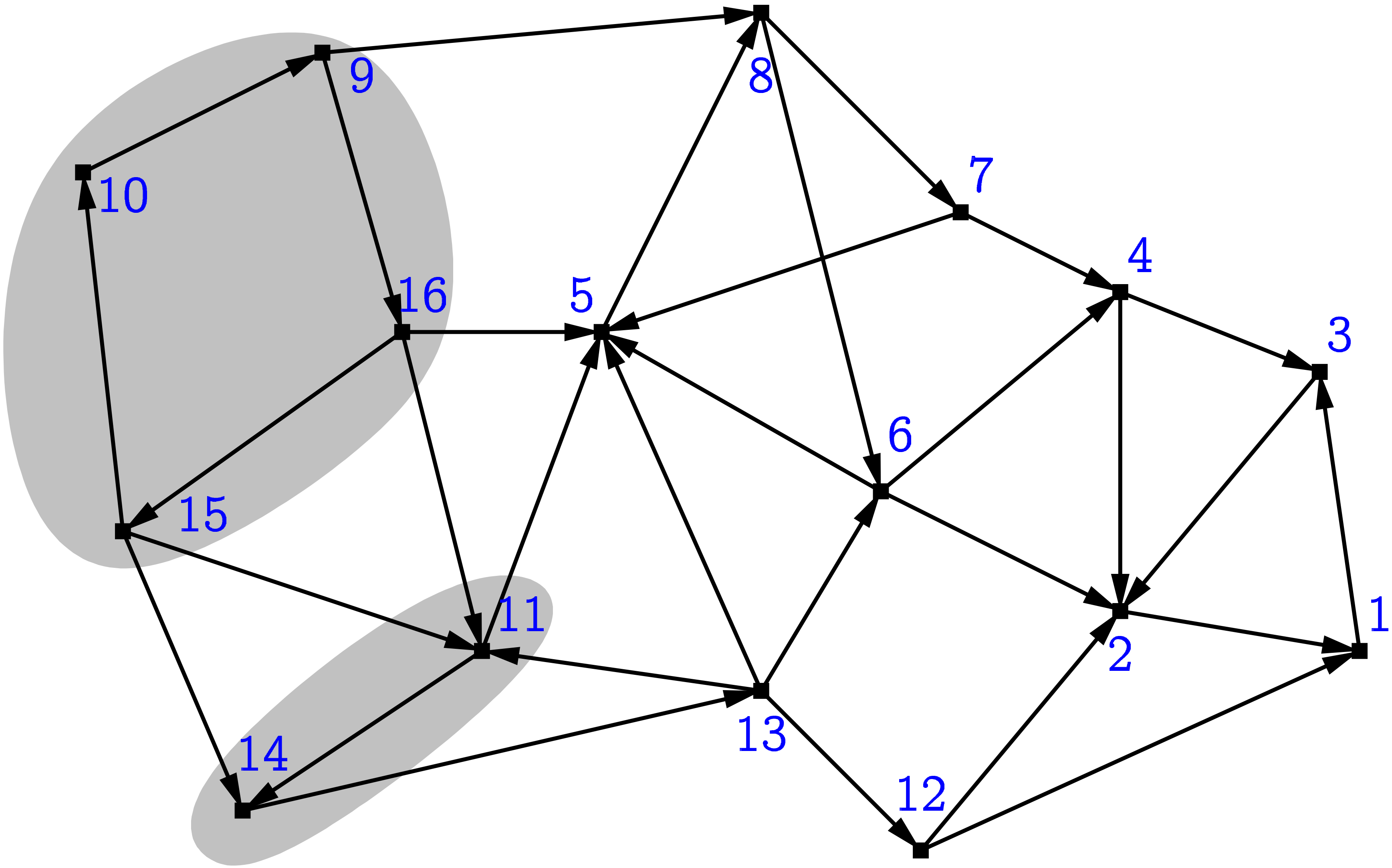
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

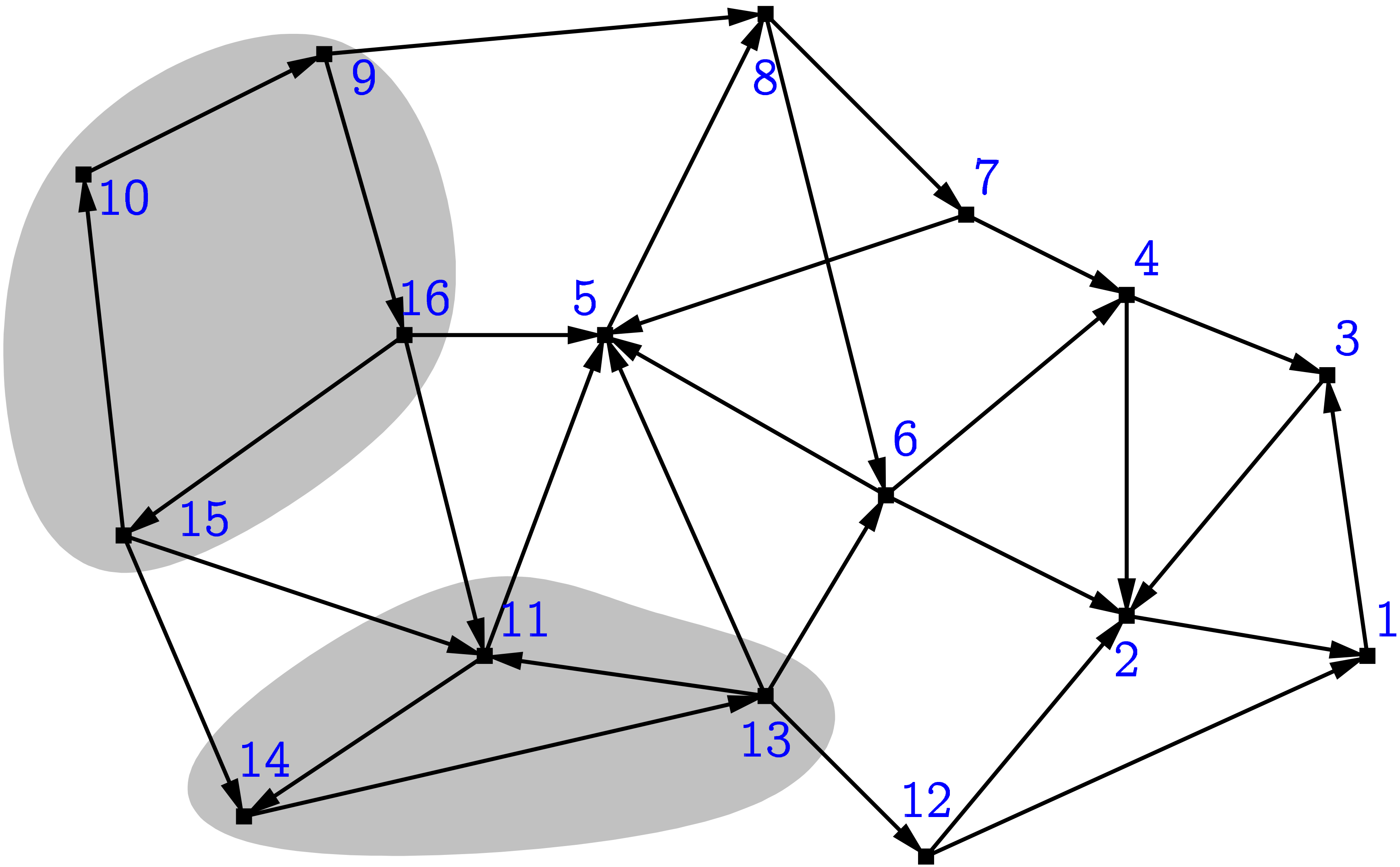


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

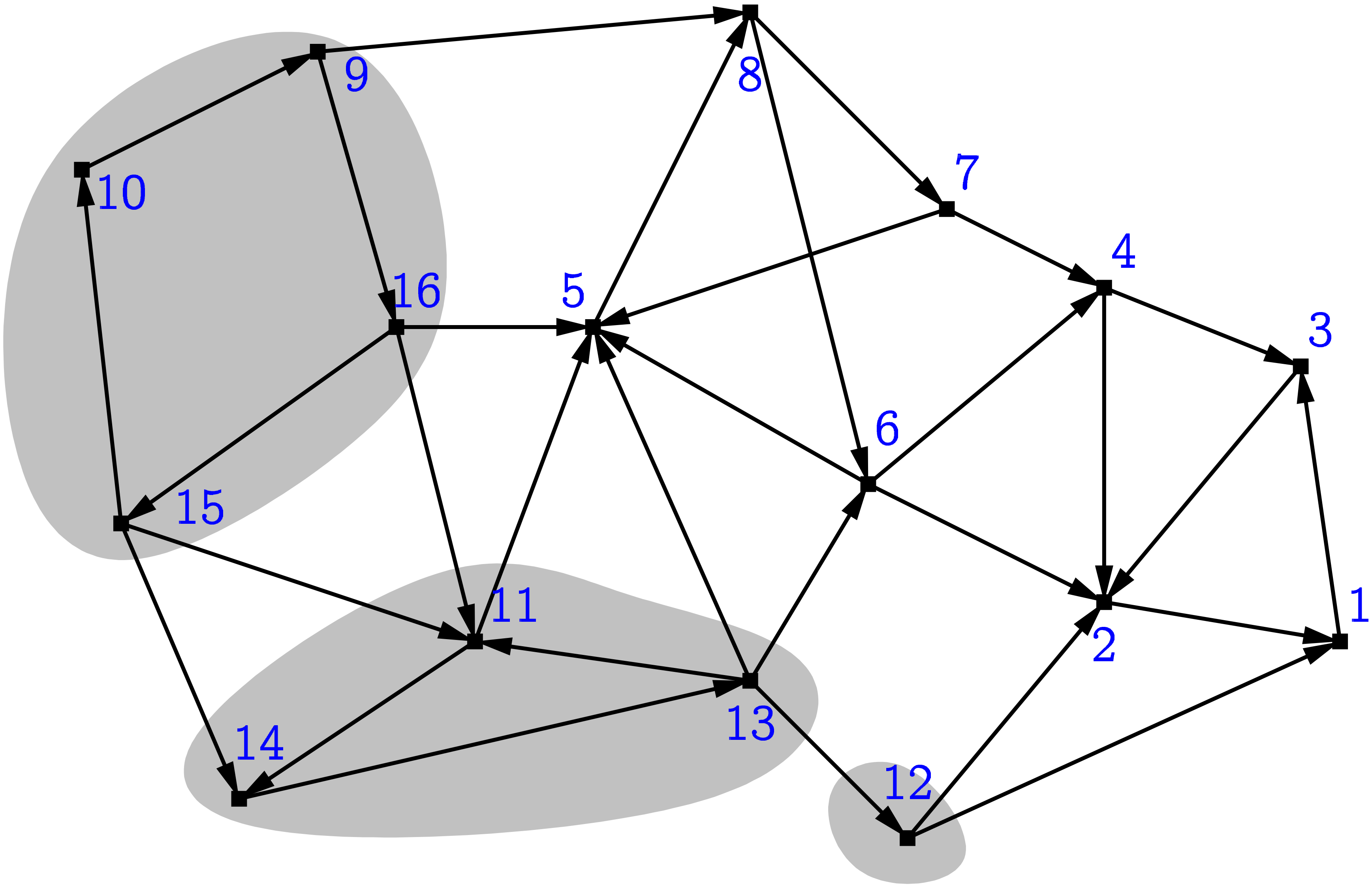


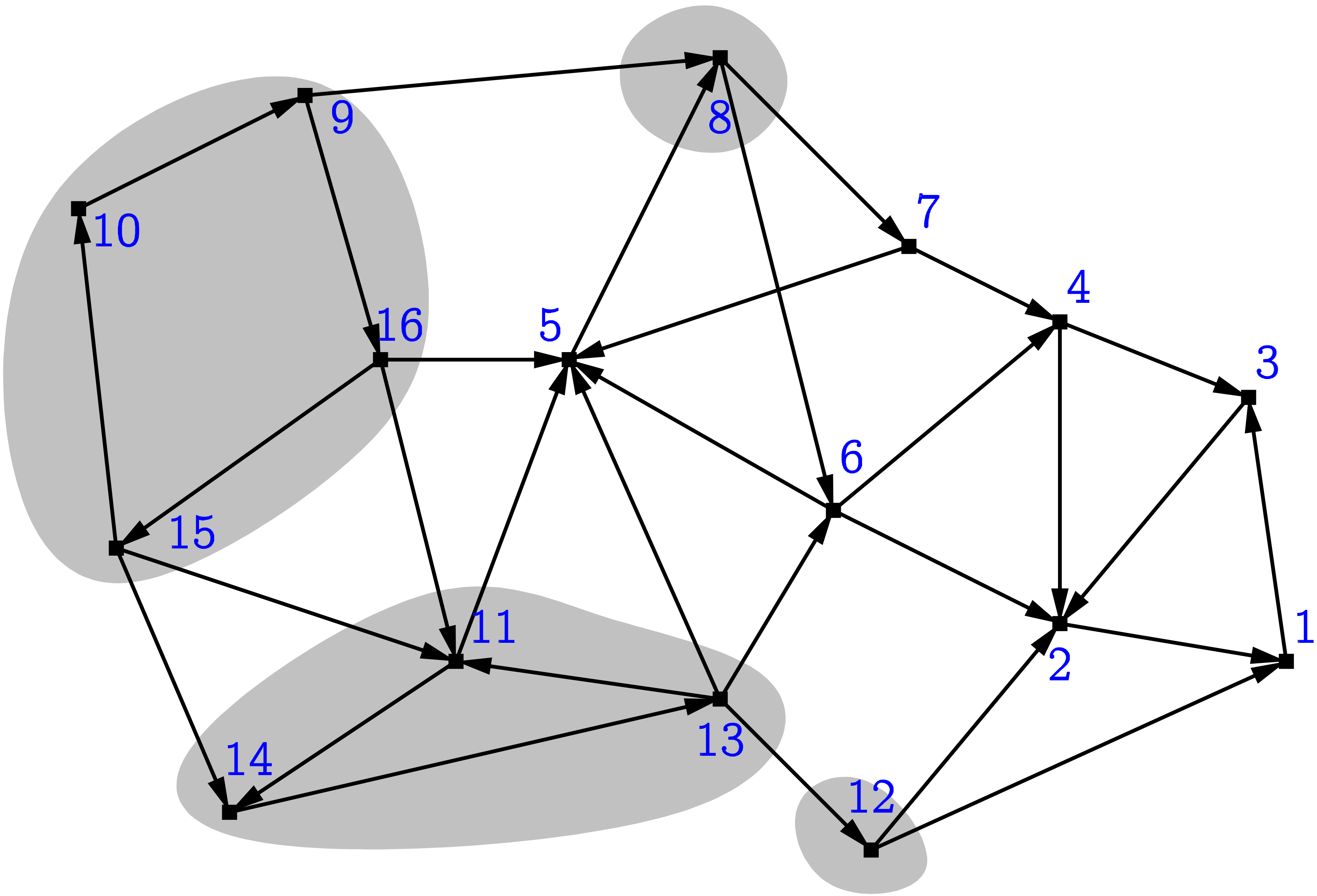
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


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

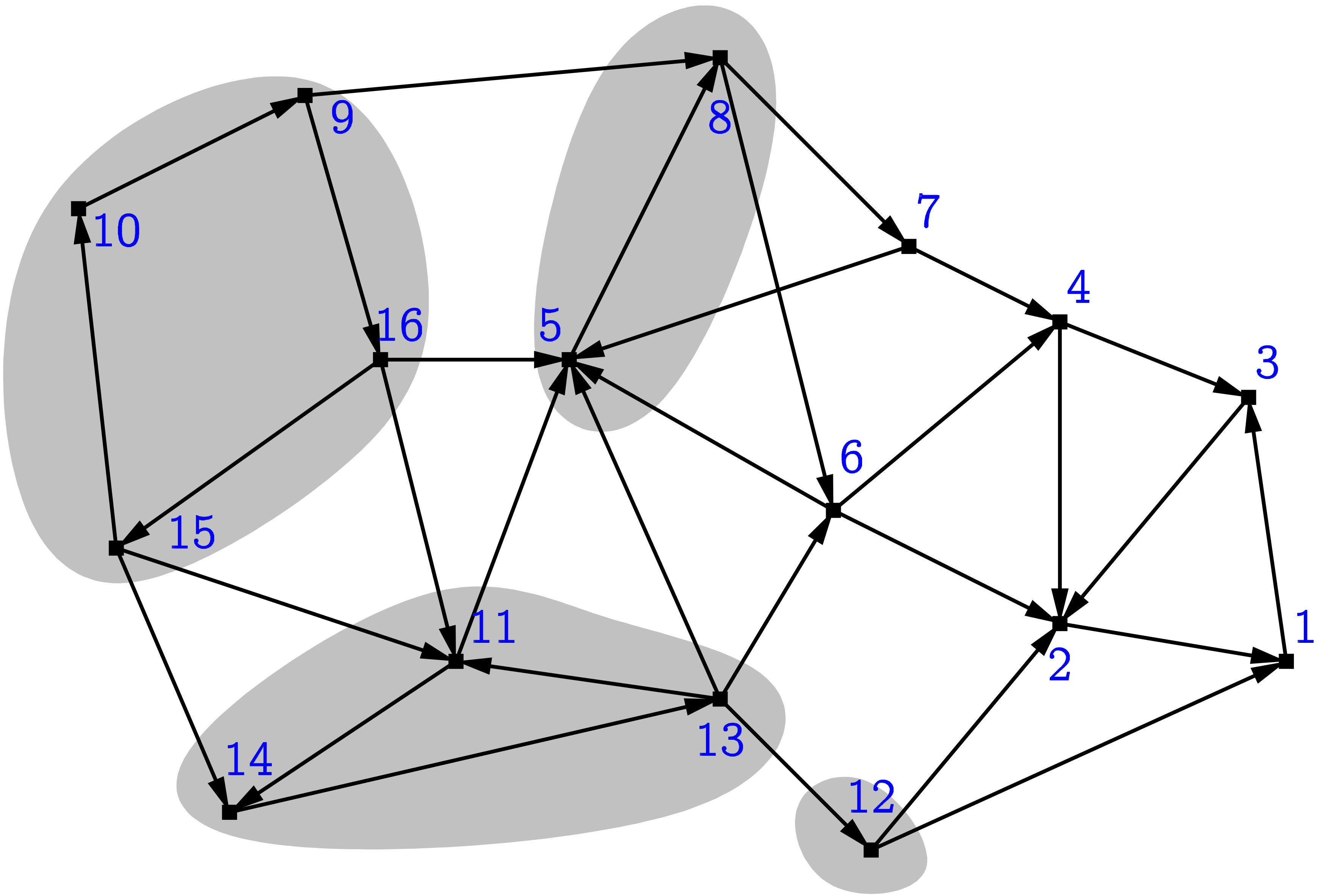


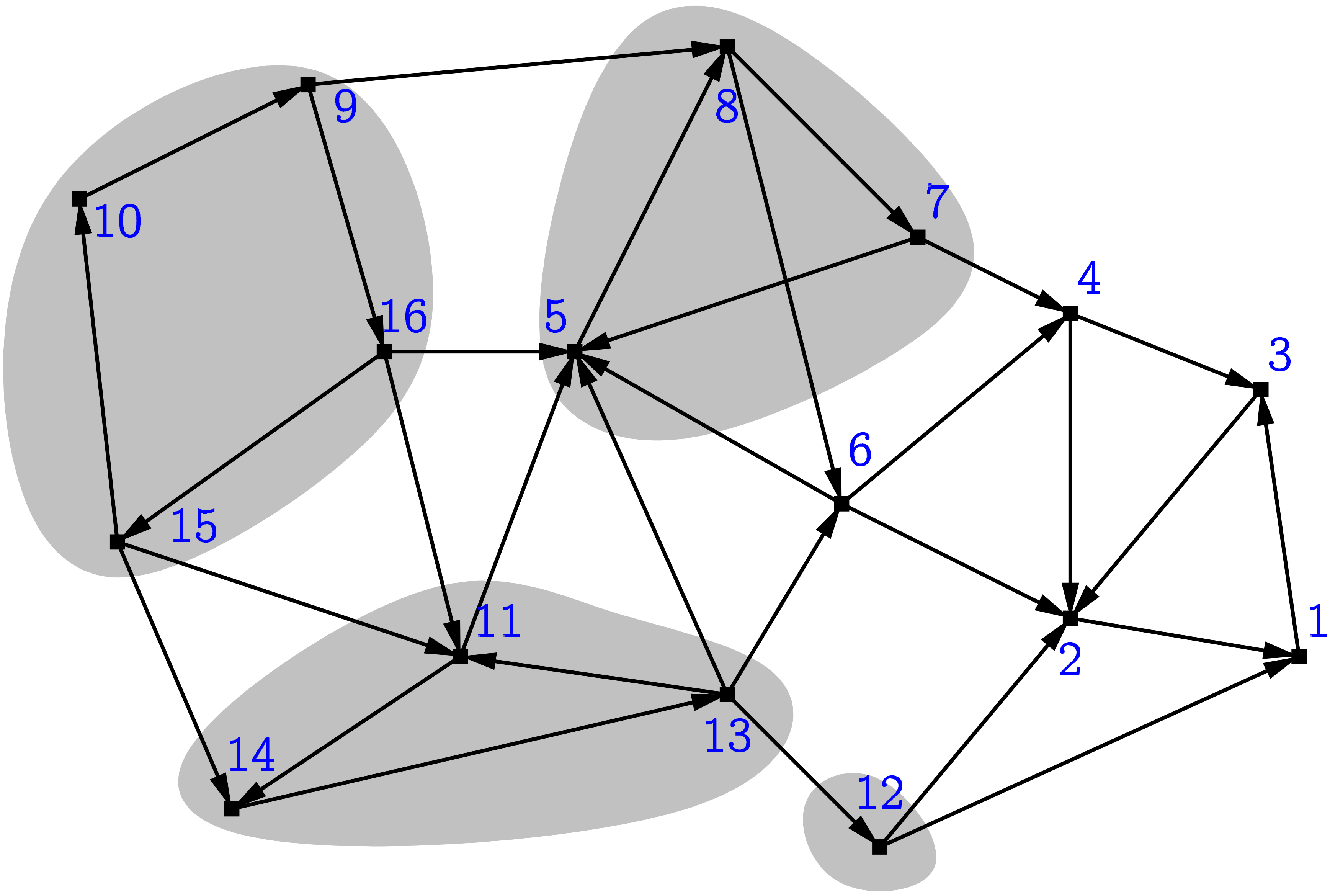
9
8
10
7
4
16
5
3
6
15
11
1
2
13
14
12


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

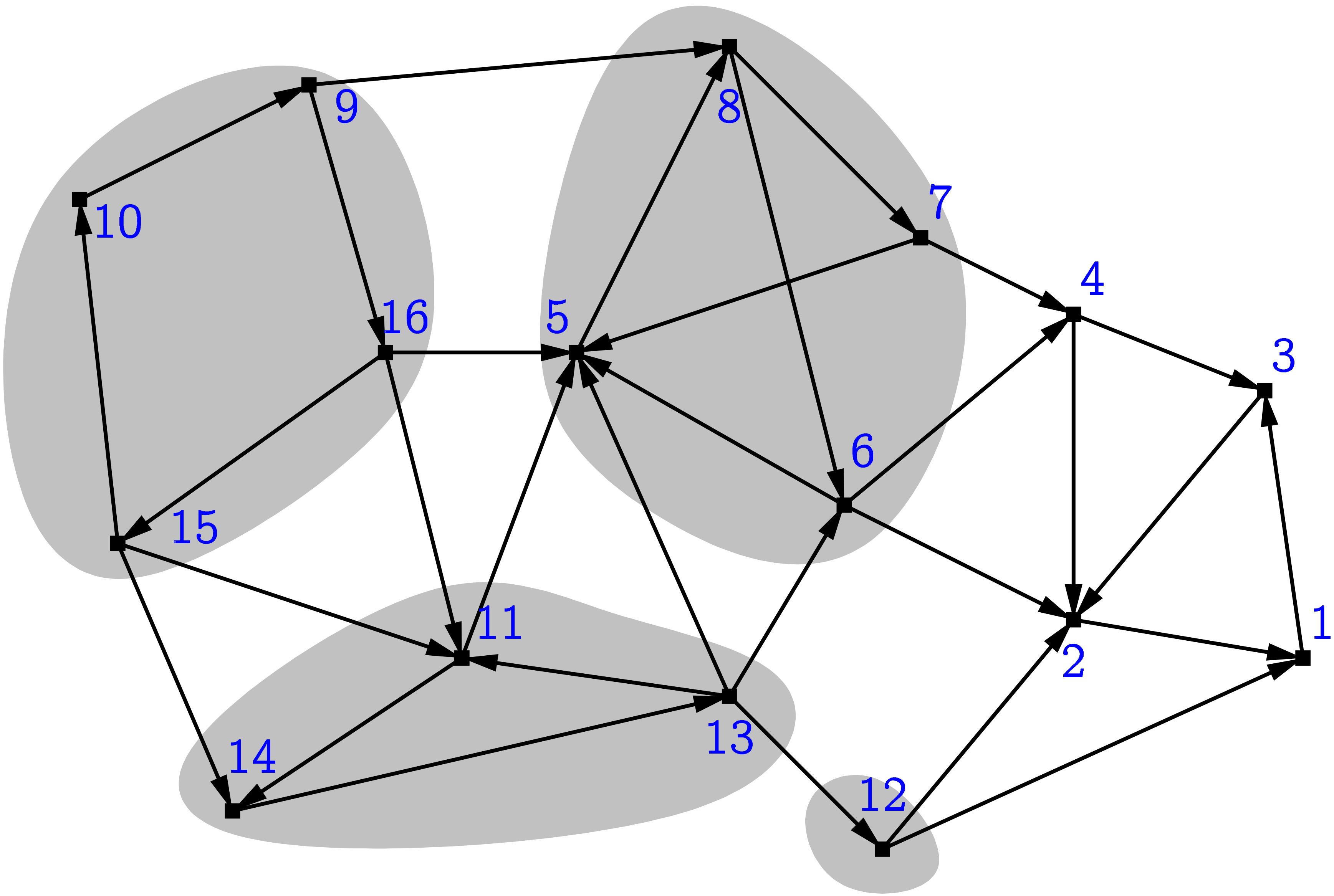


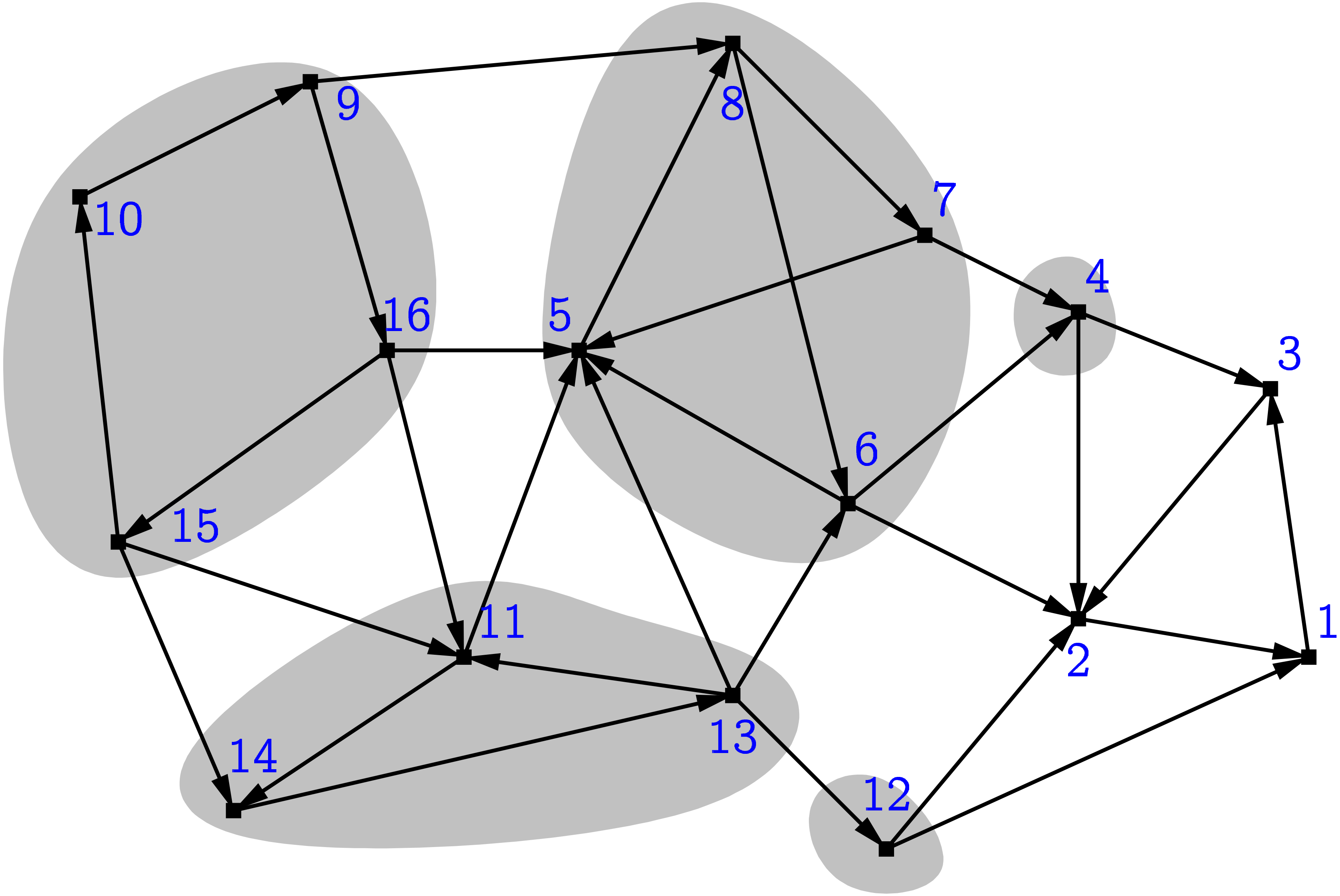
9
8
10
7
4
16
5
3
6
15
11
1
2
13
14
12

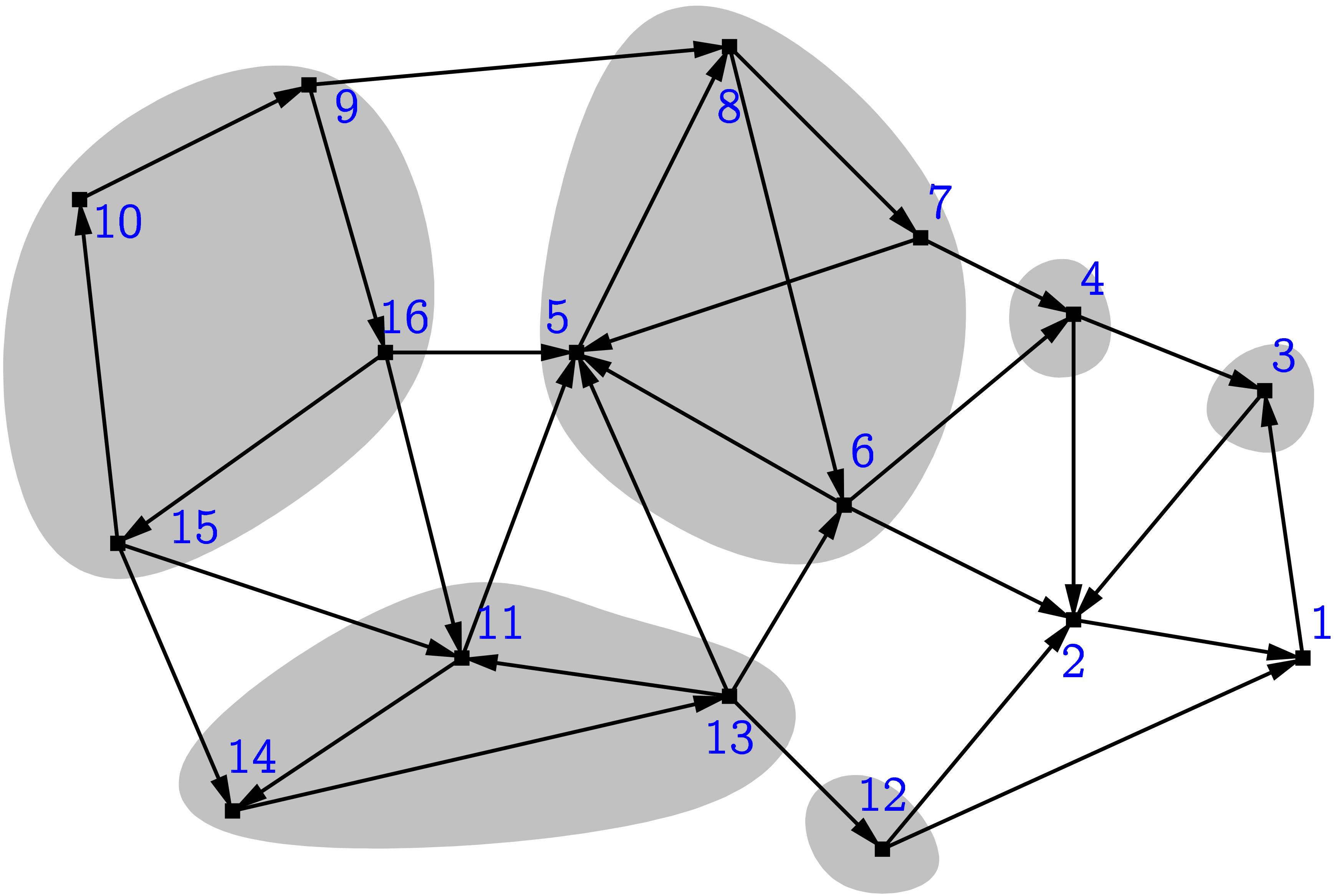

9
8
10
7
4
16
5
3
6
15
11
1
2
13
14
12

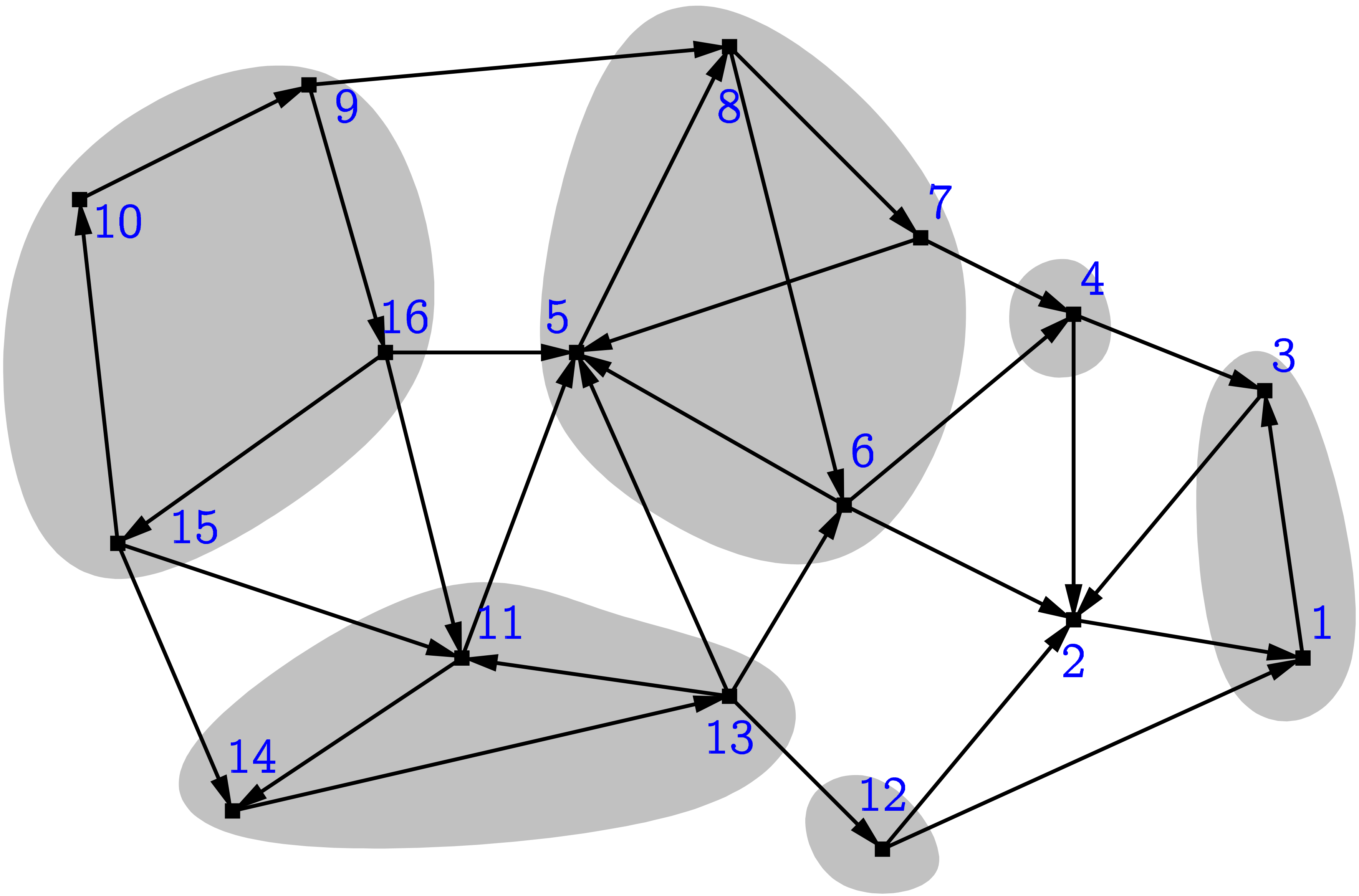


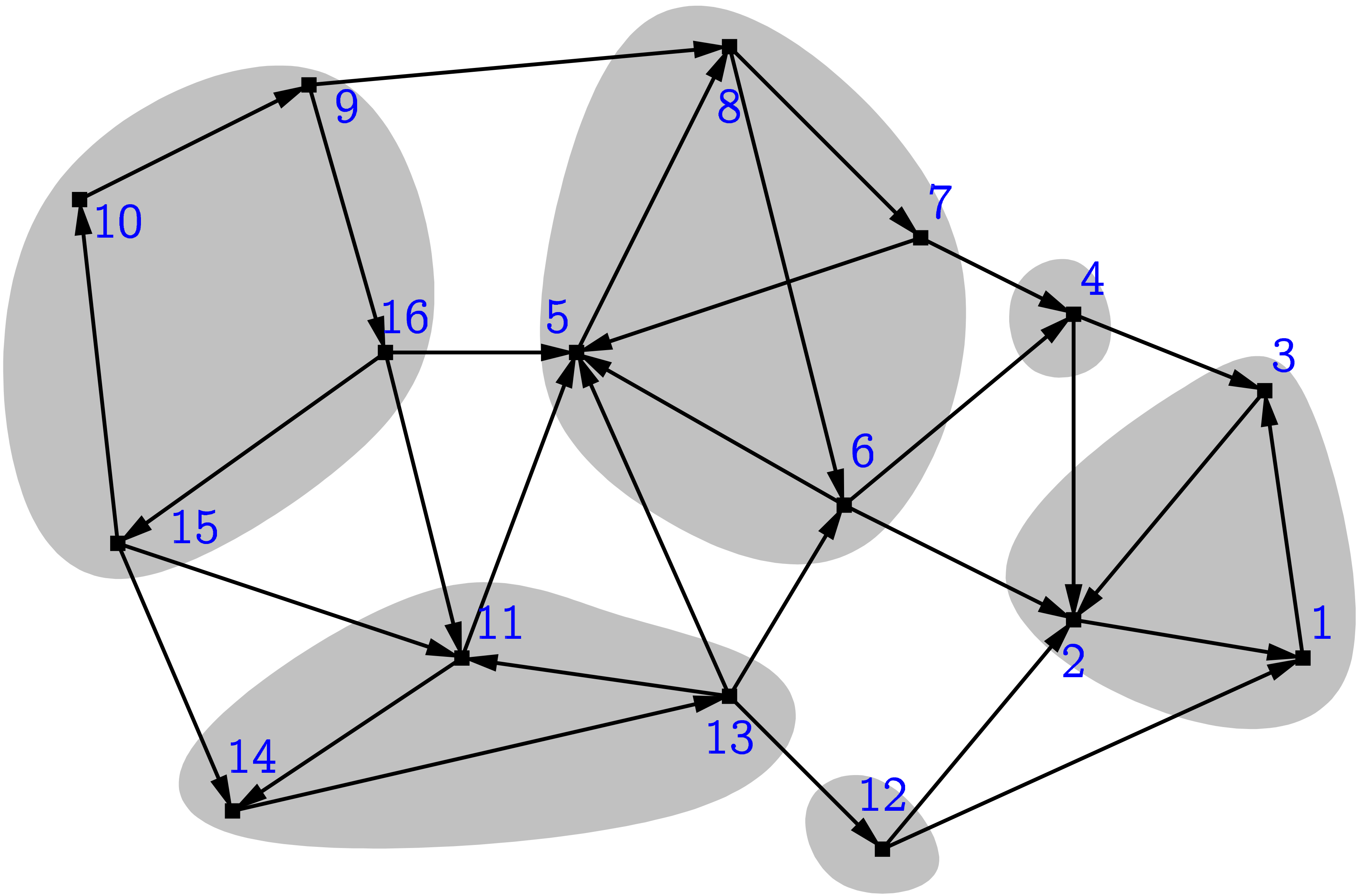
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


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


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


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


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



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

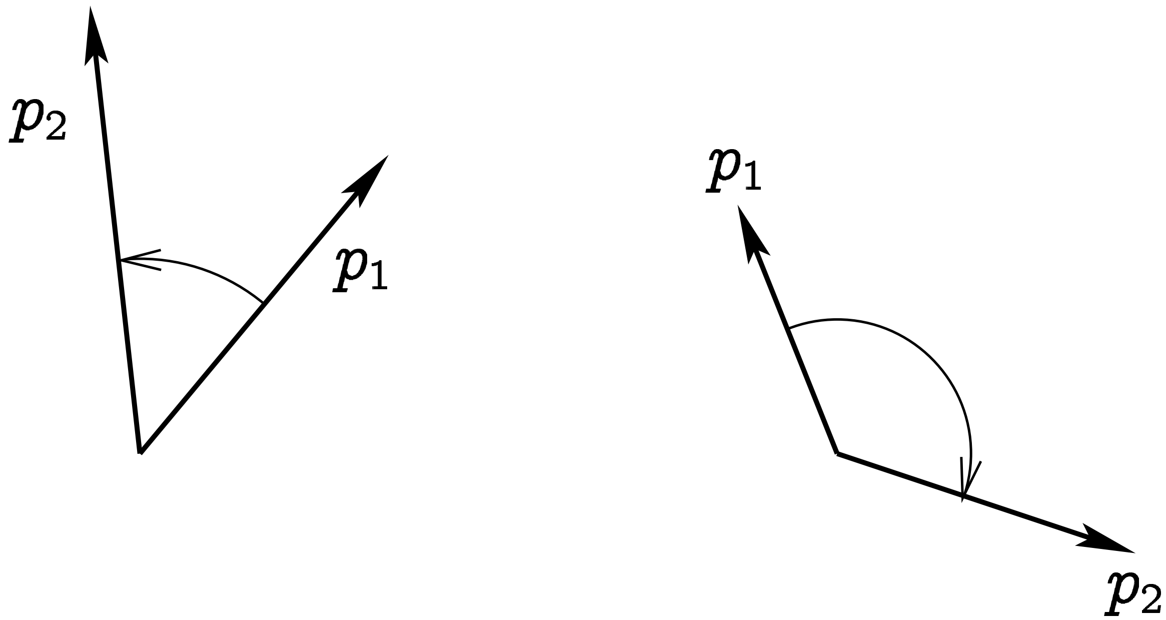
Tasandi punktide hulk $\cong \mathbb{R} \times \mathbb{R}$.

(Sirg)lõiku kujutame tema otspunktide paarina.

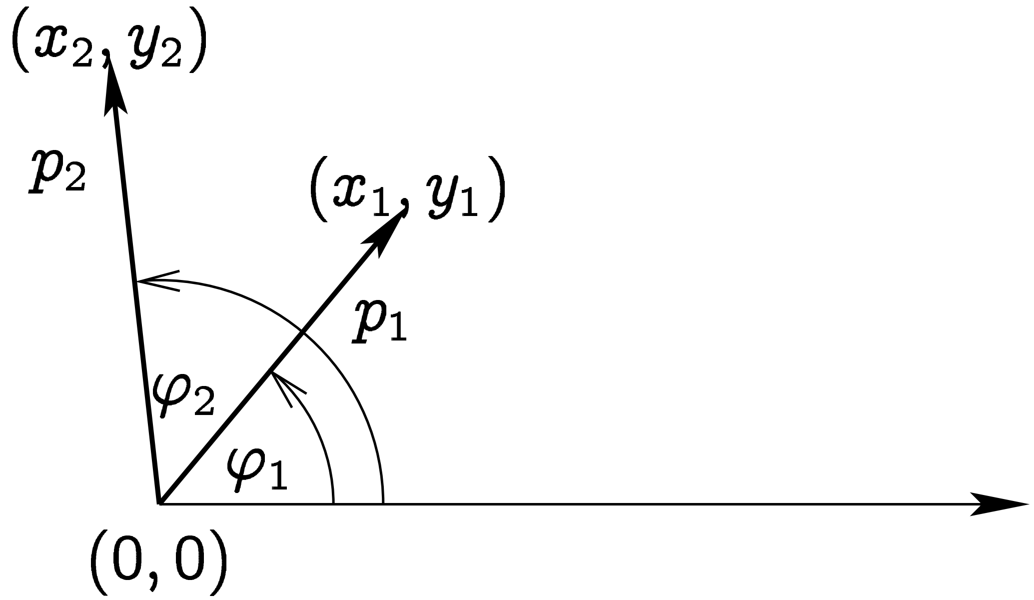
Vektorit kujutame ühe punktina — sellena, milleks ta teisendab koordinaatide alguspunkti.

Antud n sirglõiku. Kas nende seas leidub kaks tükki, mis teineteisega lõikuvad?

Antud kaks vektorit p_1 ja p_2. Vektorit p_1 tuleb pöörata ülimalt 180° ühes või teises suunas, selleks, et ta langeks kokku vektoriga p_2.



Kummas suunas?



Meid huvitab $\varphi_2 - \varphi_1$. Meid huvitab, kas see vahe on 180°-st suurem või väiksem.

Meid huvitab, kas $\sin(\varphi_2 - \varphi_1)$ on positiivne või negatiivne.

$$\sin(\varphi_2 - \varphi_1) = \sin\varphi_2 \cos\varphi_1 - \cos\varphi_2 \sin\varphi_1 = \frac{y_2}{|p_2|} \cdot \frac{x_1}{|p_1|} - \frac{x_2}{|p_2|} \cdot \frac{y_1}{|p_1|} \quad .$$

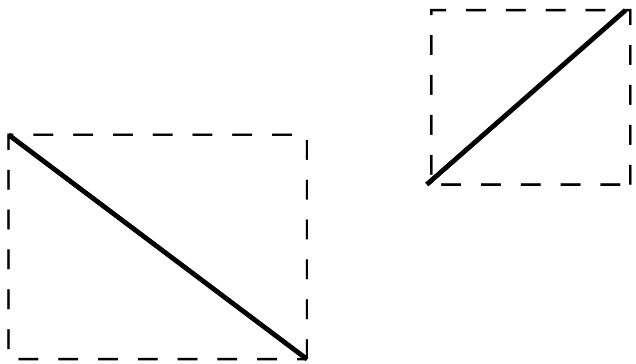
Kuna $|p_1|$ ja $|p_2|$ on positiivsed, siis huvitab meid avaldise $x_1 y_2 - x_2 y_1$ märk.

Kui see on positiivne, siis on p_2 p_1-st „vasakul", kui negatiivne, siis „paremal". Kui ta on 0, siis on p_1 ja p_2 samasihilised.

Kuidas kontrollida, kas lõigud $\overline{p_1 p_2}$ ja $\overline{p_3 p_4}$ lõikuvad?

Ebatäpsusi sissetoovaid tehteid (näiteks jagamine) ei tahaks seejuures kasutada.

Kontrollime kõigepealt, kas neid piiravad horisontaalsete ja vertikaalsete külgedega ristkülikud lõikuvad.



Kui ei, siis ei lõiku ka sirglõigud.

Olgu $p_i = (x_i, y_i)$ (kus $1 \leqslant i \leqslant 4$). Olgu

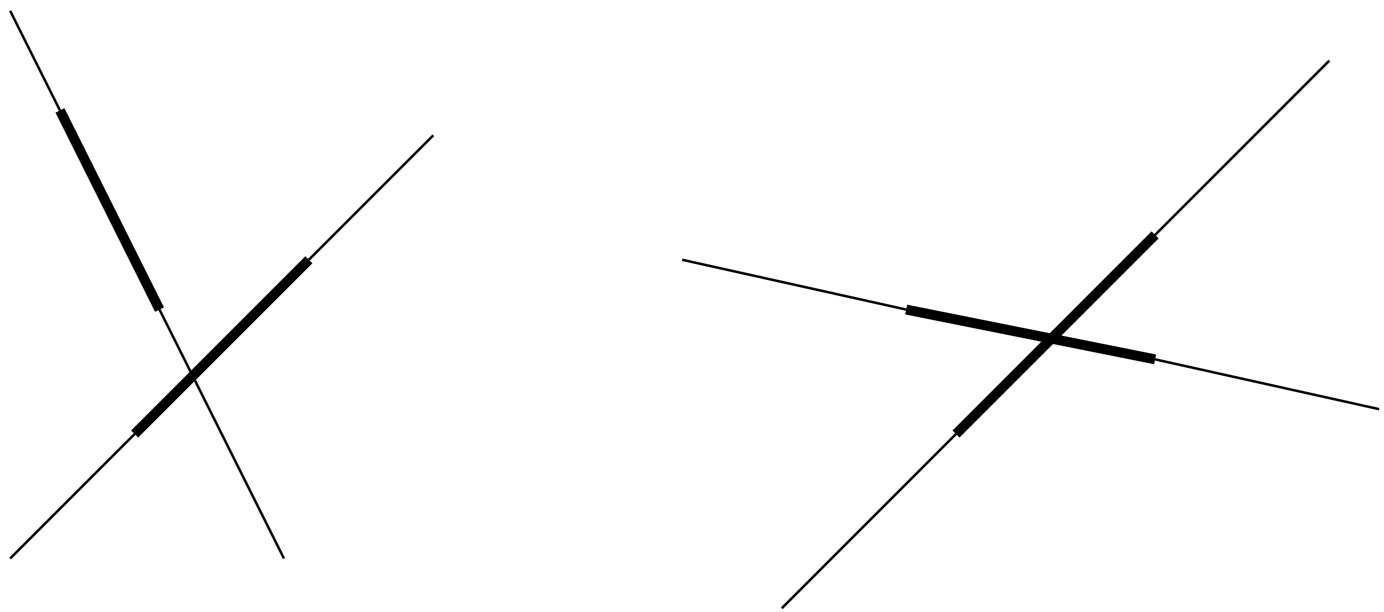
$$\begin{aligned} \hat{x}_1 &= \min(x_1, x_2) & \hat{y}_1 &= \min(y_1, y_2) \\ \hat{x}_2 &= \max(x_1, x_2) & \hat{y}_2 &= \max(y_1, y_2) \\ \hat{x}_3 &= \min(x_3, x_4) & \hat{y}_3 &= \min(y_3, y_4) \\ \hat{x}_4 &= \max(x_3, x_4) & \hat{y}_4 &= \max(y_3, y_4), \end{aligned}$$

Siis need ristkülikudd on vastavalt $\hat{x}_1 \leqslant x \leqslant \hat{x}_2$, $\hat{y}_1 \leqslant y \leqslant \hat{y}_2$ ja $\hat{x}_3 \leqslant x \leqslant \hat{x}_4$, $\hat{y}_3 \leqslant y \leqslant \hat{y}_4$. Nad lõikuvad parajasti siis, kui nad lõikuvad mõlemas dimensioonis.

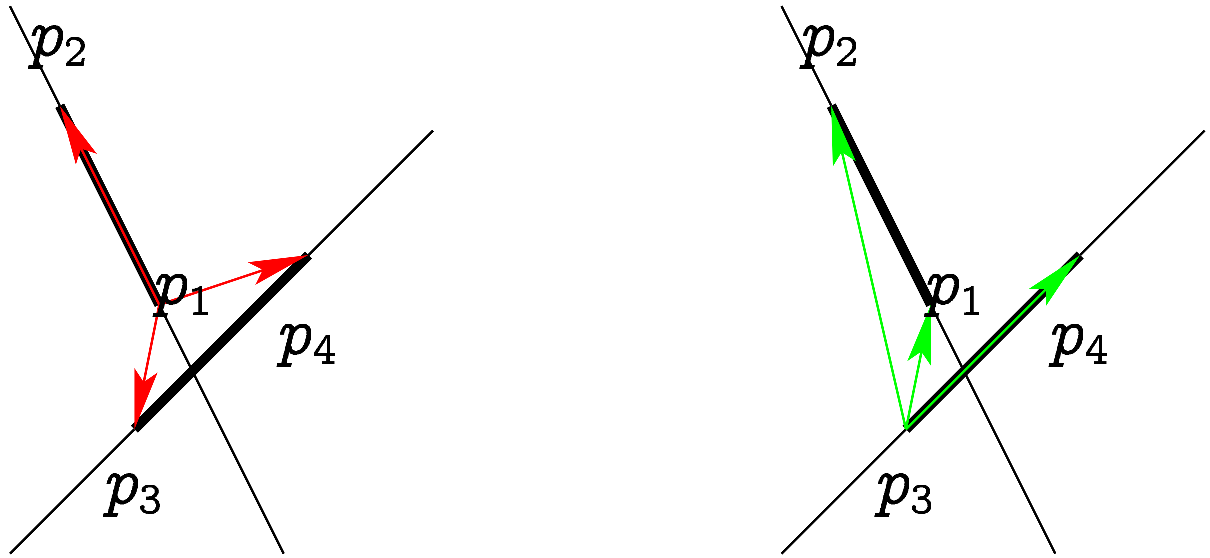
Lõikuvad, kui $z_3 \leqslant z_2$ ja $z_1 \leqslant z_4$.

Oletame, et piiravad ristkülikud lõikuvad. Vaatame sirgeid, mille määravad need sirglõigud.

Kui üks lõik asub teisega määratud sirgest ühel pool, siis nad ei lõiku. Kui kumbki lõikab teisega määratud sirget, siis nad lõikuvad.



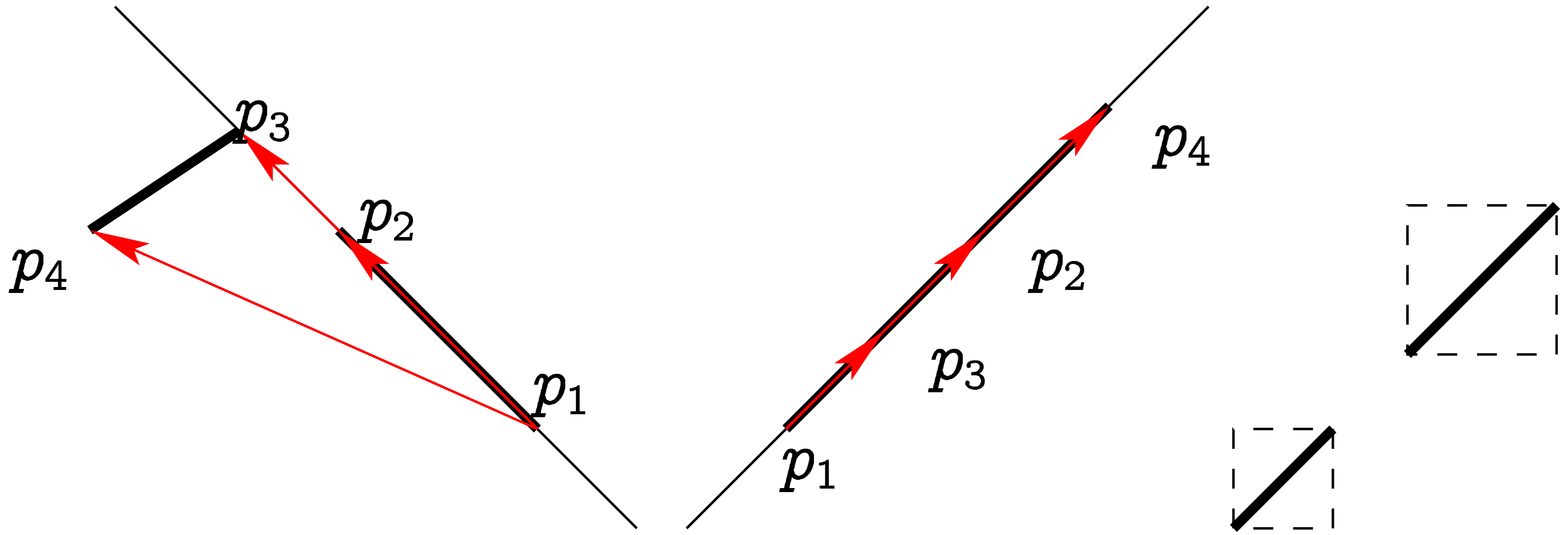
$\overline{p_3p_4}$ asub $\overline{p_1p_2}$-ga määratud sirgest ühel pool parajasti siis, kui tema mõlemad otspunktid asuvad sellest ühel pool.



Meil tuleb leida, kummale poole jäävad vektorist $\overrightarrow{p_1p_2}$ vektorid $\overrightarrow{p_1p_3}$ ja $\overrightarrow{p_1p_4}$.

Samuti, kummale poole jäävad vektorist $\overrightarrow{p_3p_4}$ vektorid $\overrightarrow{p_3p_1}$ ja $\overrightarrow{p_3p_2}$.

Mis siis, kui mõned neist vektoritest on samasihilised?



Siis võime lugeda, et üks lõik lõikub teise poolt määratud sirgega.

Parempoolne variant pole siinkohal võimalik, sest me oleme juba kontrollinud, kas piiravad ristkülikud lõikuvad.

vekt_asend$((x_1, y_1), (x_2, y_2))$ on

$d := x_1 y_2 - x_2 y_1$
if $d > 0$ **then return** „+“
if $d < 0$ **then return** „−“ **else return** „0“

p3_asend$((x_1, y_1), (x_2, y_2), (x_3, y_3))$ on

return vekt_asend$((x_2 - x_1, y_2 - y_1), (x_3 - x_1, y_3 - y_1))$

lõik_sirge?$((x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4))$ on

$a :=$ p3_asend$((x_1, y_1), (x_2, y_2), (x_3, y_3))$
$b :=$ p3_asend$((x_1, y_1), (x_2, y_2), (x_4, y_4))$
if $a =$ „0“ **or** $b =$ „0“ **then return** true
return $(a \neq b)$

lõik_bbox?$((x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4))$ on

$\hat{x}_1 := \min(x_1, x_2);\ \hat{y}_1 := \min(y_1, y_2)$
$\hat{x}_2 := \max(x_1, x_2);\ \hat{y}_2 := \max(y_1, y_2)$
$\hat{x}_3 := \min(x_3, x_4);\ \hat{y}_3 := \min(y_3, y_4)$
$\hat{x}_4 := \max(x_3, x_4);\ \hat{y}_4 := \max(y_3, y_4)$
return $(\hat{x}_3 \leqslant \hat{x}_2) \wedge (\hat{x}_1 \leqslant \hat{x}_4) \wedge (\hat{y}_3 \leqslant \hat{y}_2) \wedge (\hat{y}_1 \leqslant \hat{y}_4)$

lõikuvad_lõigud?(p_1, p_2, p_3, p_4) on

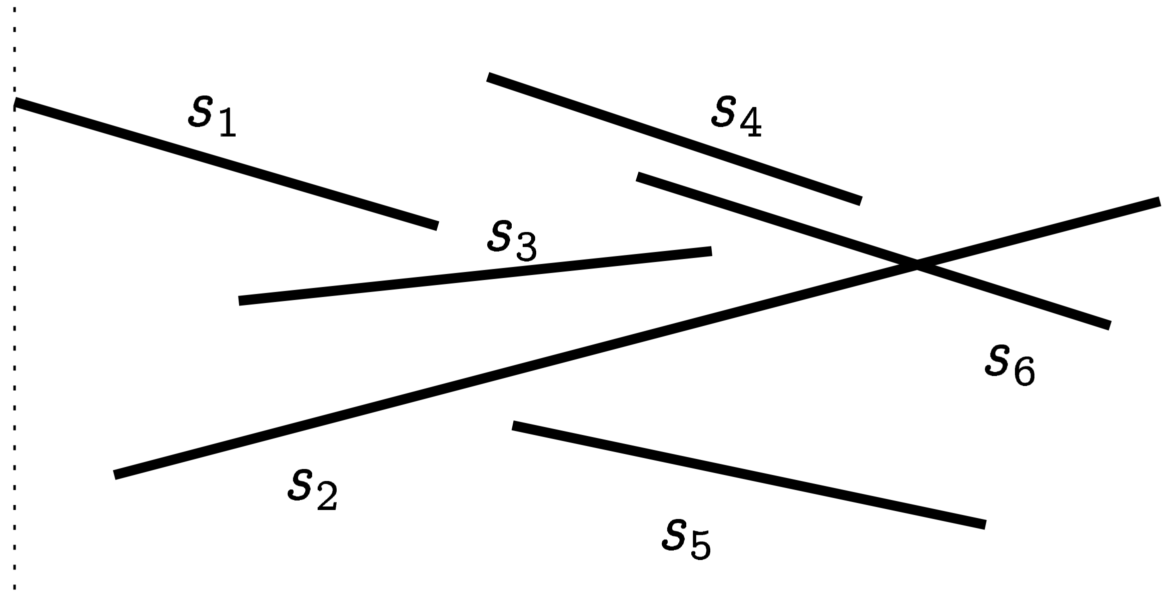
return lõik_bbox?(p_1, p_2, p_3, p_4)
  and lõik_sirge?(p_1, p_2, p_3, p_4)
  and lõik_sirge?(p_3, p_4, p_1, p_2)

Olgu meil nüüd antud sirglõigud $s_1, \ldots, s_n$. Tahame leida, kas nende seas mõned omavahel lõikuvad. Kõiki lõikuvaid paare ei taha leida.

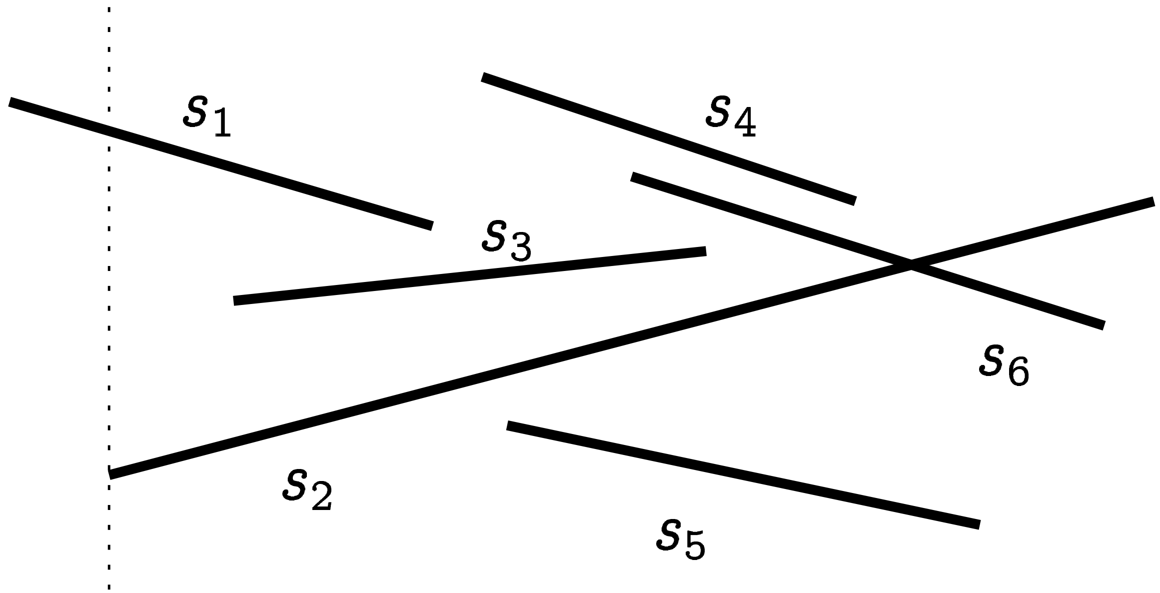
Loeme, et ükski lõik ei ole vertikaalne ning ükski kolmik ei lõiku ühes punktis.

Lahendusidee: libistame vertikaalset joont üle nende lõikude vasakult paremale, kuni leiame lõikepunkti (või jõuame kõigist lõikudest üle).

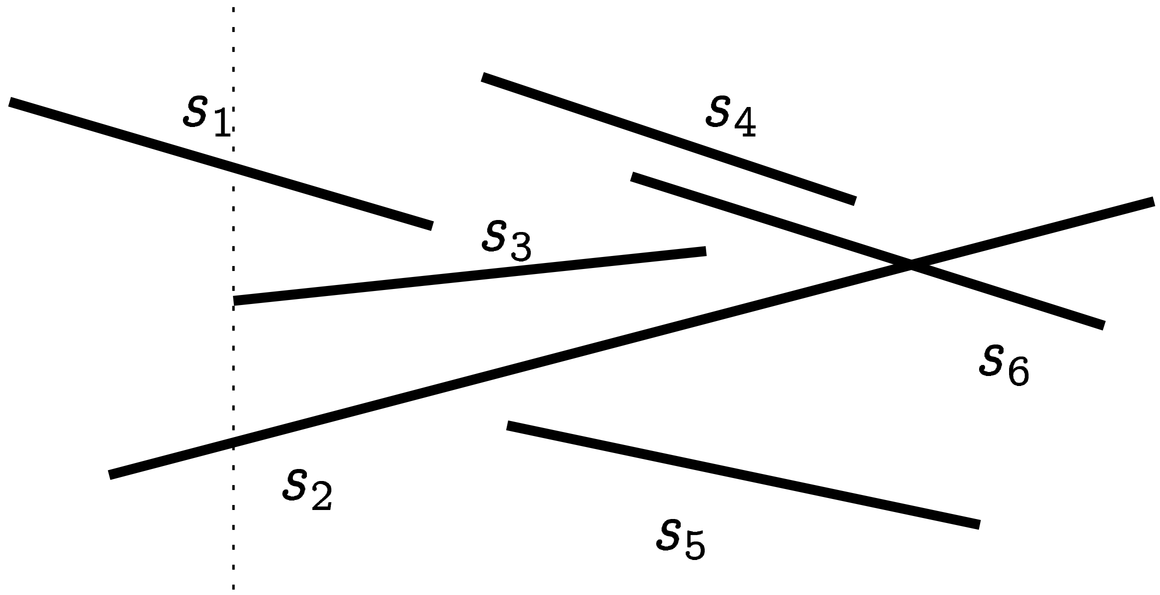
Peame arvet, mis järjekorras antud sirglõigud selle vertikaalse joonega lõikuvad.



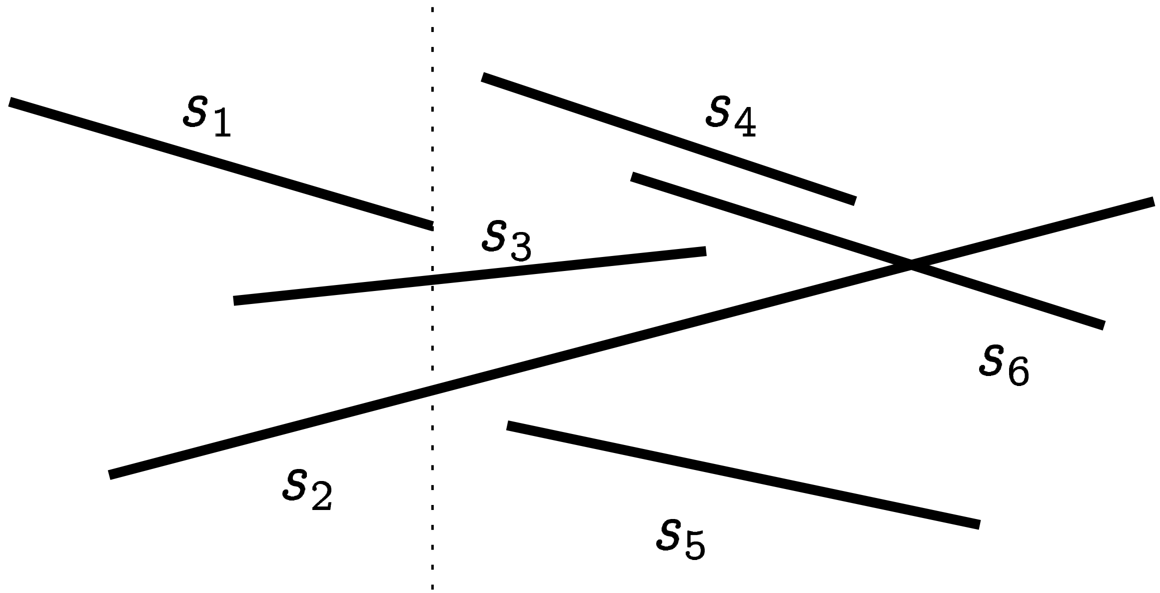
Lõigatavad lõigud: s_1



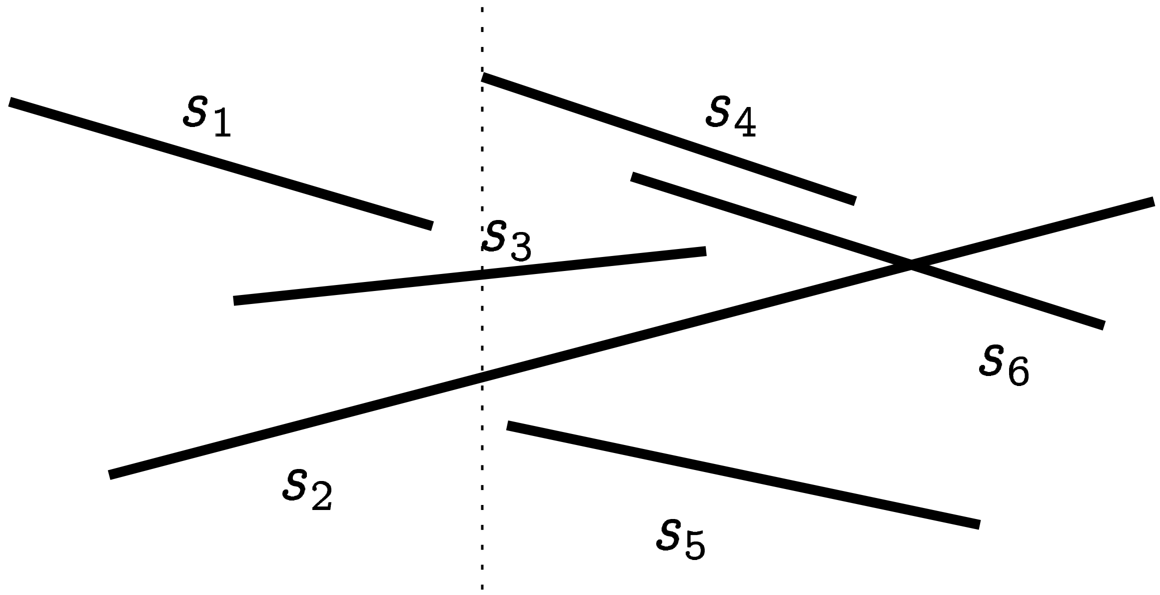
Lõigatavad lõigud: s_1, s_2



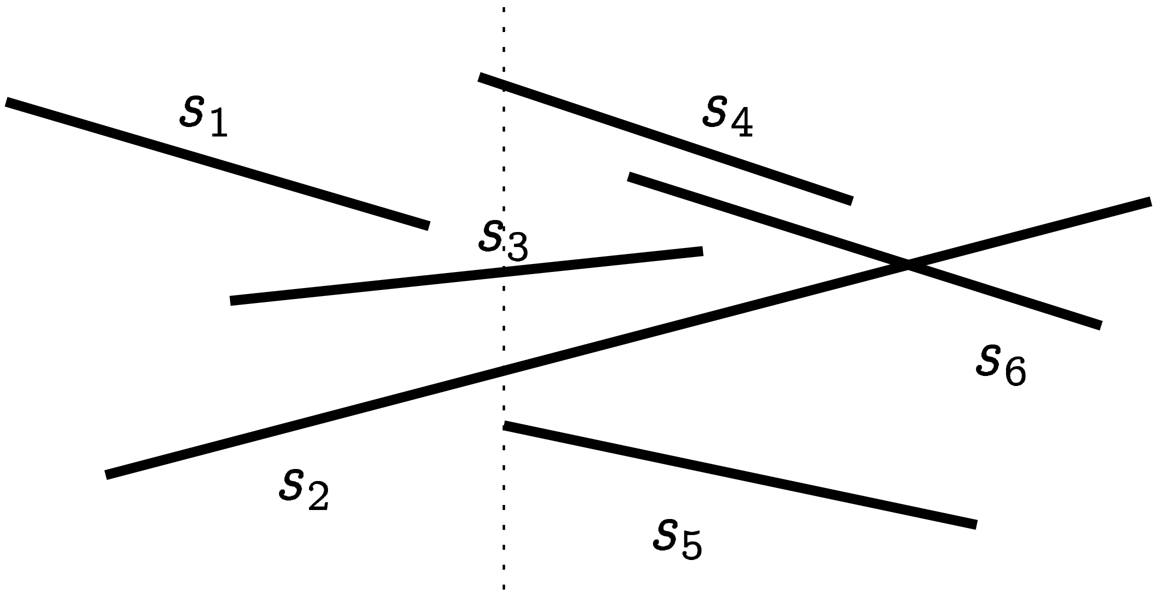
Lõigatavad lõigud: s_1, s_3, s_2



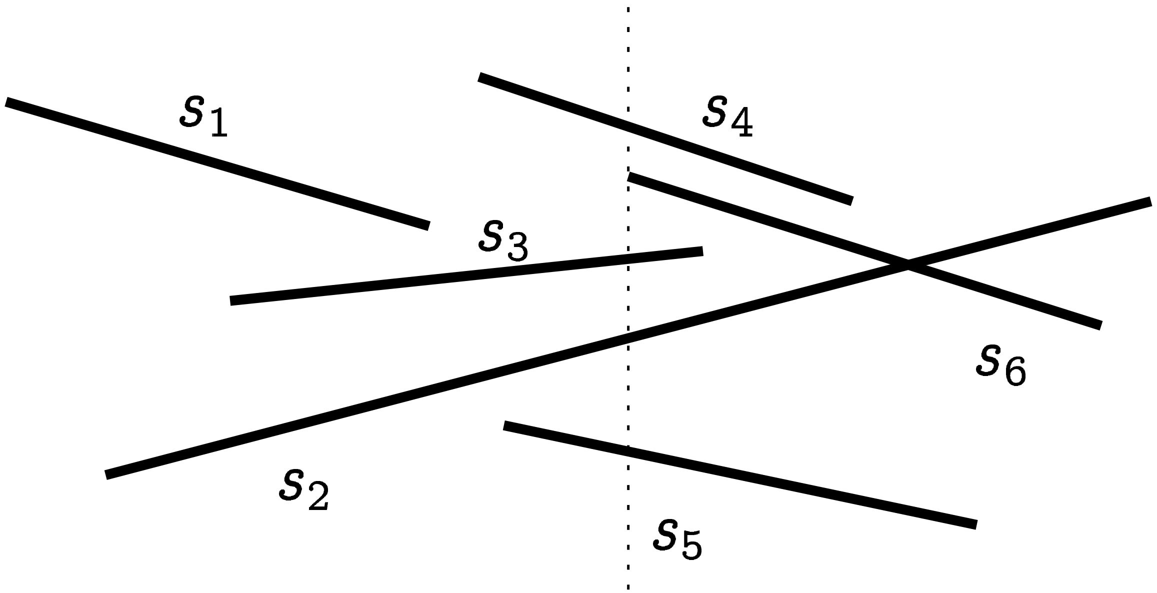
Lõigatavad lõigud: s_3, s_2



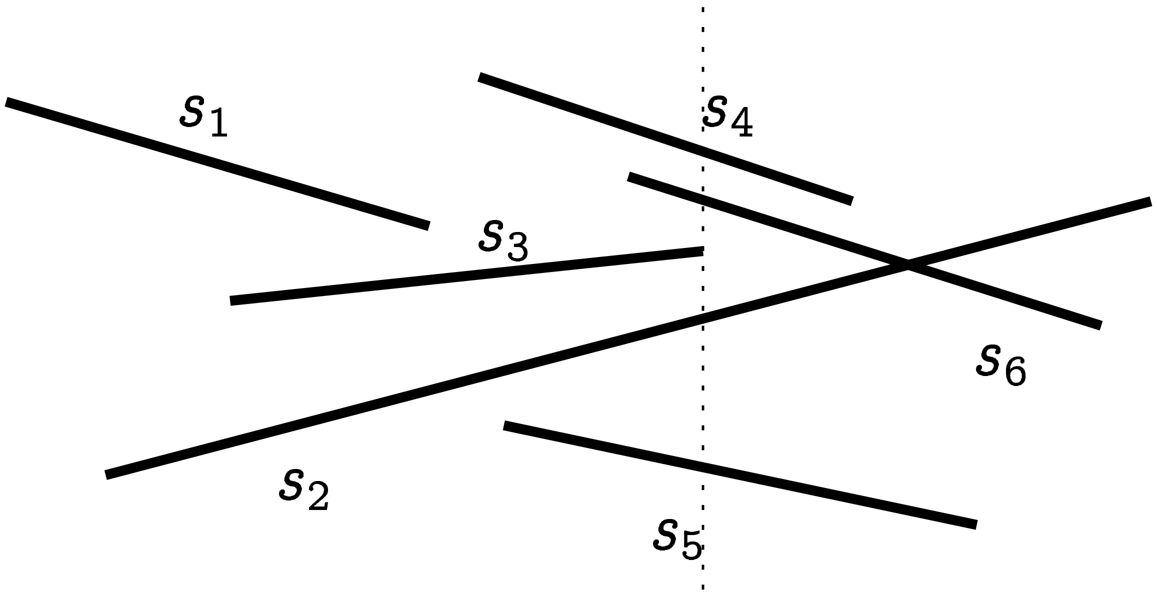
Lõigatavad lõigud: s_4, s_3, s_2



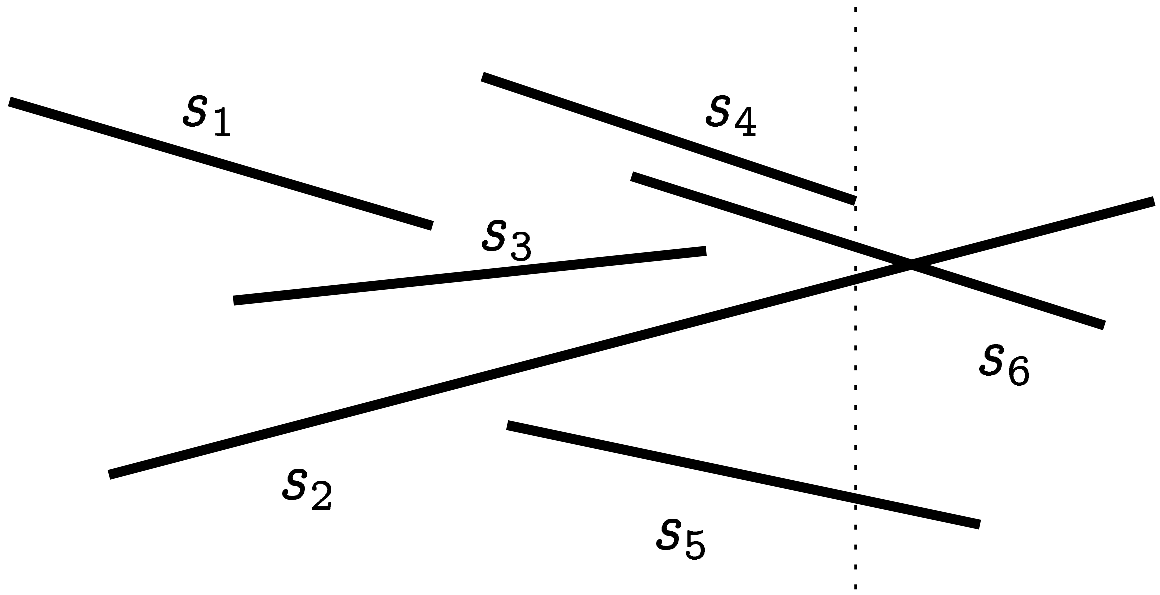
Lõigatavad lõigud: s_4, s_3, s_2, s_5



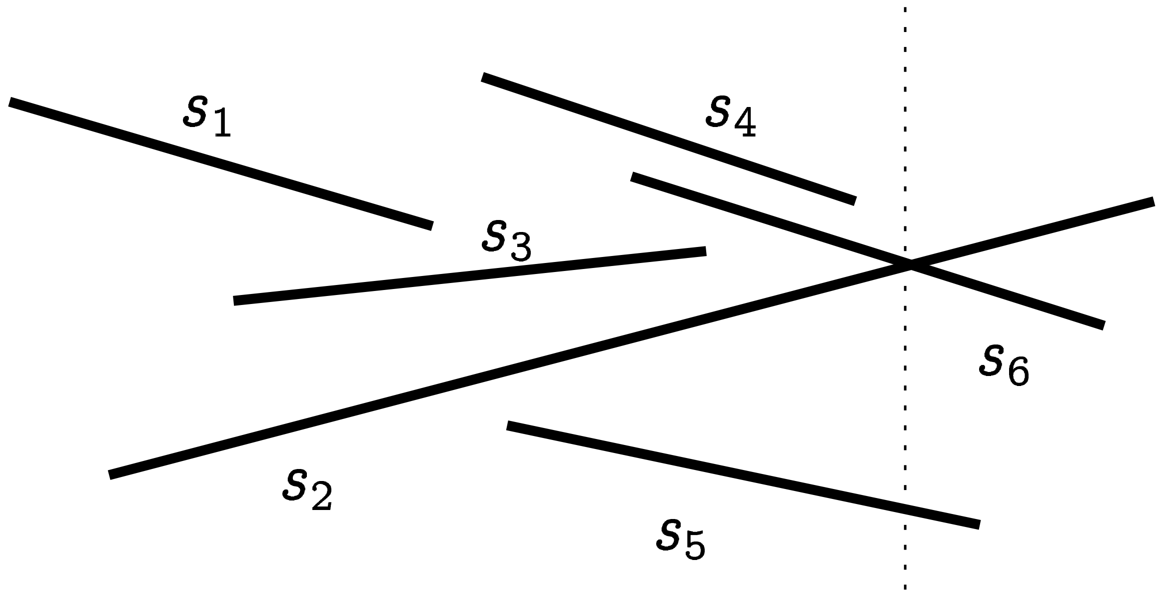
Lõigatavad lõigud: s_4, s_6, s_3, s_2, s_5



Lõigatavad lõigud: s_4, s_6, s_2, s_5



Lõigatavad lõigud: s_6, s_2, s_5



Lõigatavad lõigud: s_2, s_6, s_5

Paneme tähele, et senikaua, kuni pole olnud ühtegi lõikepunkti, lõikude suhteline asend vertikaalsel joonel ei muutu.

Enne, kui mingid kaks lõiku lõikuvad, peavad nad vertikaalsel joonel järjest esinema.

Algoritmis me võtame mingite kohtade peal vertikaalsed jooned, nendel järjest mingid kaks lõiku ning kontrollime, kas need lõigud lõikuvad.

Kohtade valik on selline, et me ühtegi lõikumist maha ei maga.

Olgu $p_1, \ldots, p_{2n}$ lõikude $s_1, \ldots, s_n$ otspunktid, mis on sorteeritud x-koordinaadi järgi (kui kahe punkti x-koordinaadid on võrdsed, siis y-koordinaadi järgi).

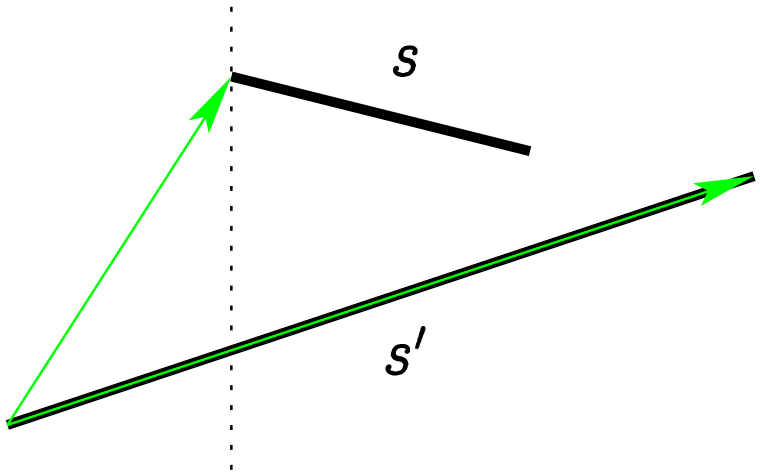
Olgu T dünaamiline järjend, mis toetab järgmisi operatsioone:

- lisa(T, s),
- kustuta(T, s),
- eelmine(T, s),
- järgmine(T, s).

T-s hoiame lõike. T-ks sobib kahendpuu. Kui ta on AVL-puu, siis on kõik operatsioonid keerukusega $O(\log n)$.

lisa(T, s) peab panema s-i kohta, mis vastab tema positsioonile vertikaalsel joonel.

Võrdlemaks, kas s asub juba T-sse kuuluvast s'-st üleval või allpool, uurime, kumba pidi asuvad allpool kujutatud vektorid.



Käi punktid $p_1, \ldots, p_{2n}$ vasakult paremale läbi ja

- Kui p_i on mingi s-i vasakpoolne otspunkt, siis
  - lisa(T, s)
  - kui eelmine(T, s) / järgmine(T, s) leidub, siis kontrolli, kas ta lõikub s-ga. Kui jah, siis tagasta „jah“ ja lõpeta.
- Kui p_i on mingi s-i parempoolne otspunkt, siis
  - kui eelmine(T, s) ja järgmine(T, s) mõlemad leiduvad, siis kontrolli, kas nad lõikuvad. Kui jah, siis tagasta „jah“ ja lõpeta.
  - kustuta(T, s).

Kui lõikumisi ei leitud, tagasta „ei“.

Ilmne on, et kui algoritm tagastab „jah“, siis leiduvad lõigud, mis lõikuvad. Algoritm on siis just ühe paari leidnud.

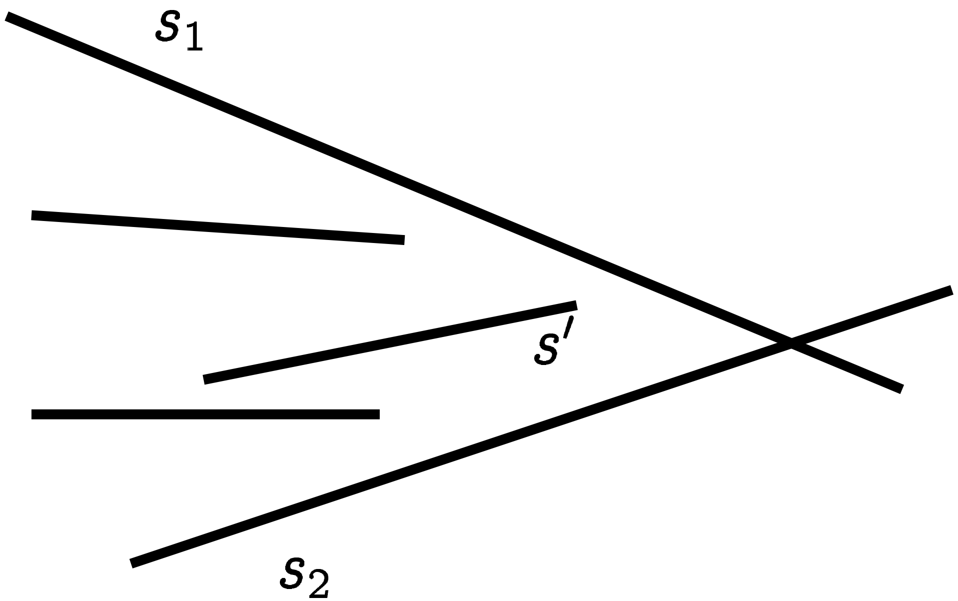
Näitame, et kui ta tagastab „ei“, siis selliseid lõike ei leidu.

Oletame vastuväiteliselt, et algoritm tagastab „ei“, aga mingid lõigud s_1 ja s_2 lõikuvad.

Lõikugu s_1 ja s_2 nii vasakul kui võimalik. Kui vähima x-koordinaadiga lõikepunkte on mitu, siis valime neist alumise.

S.t. s_1 ja s_2 lõikepunktist vasakul ei toimu ühtegi lõikumist.

Alaku s_1 enne s_2-e. Kui s_2 alguspunkti töötlemisel ei leita üles tema lõikumist s_1-ga, siis peavad mingid lõigud veel s_1 ja s_2 vahel asuma.



Vaatame neid lõike. Nad kõik lõppevad enne s_1 ja s_2 lõikumist. Olgu s' lõik, mis neist kõige hiljem lõppeb. Tema lõppemisel avastatakse s_1 ja s_2 lõikumine.

Algoritmi tööaeg:

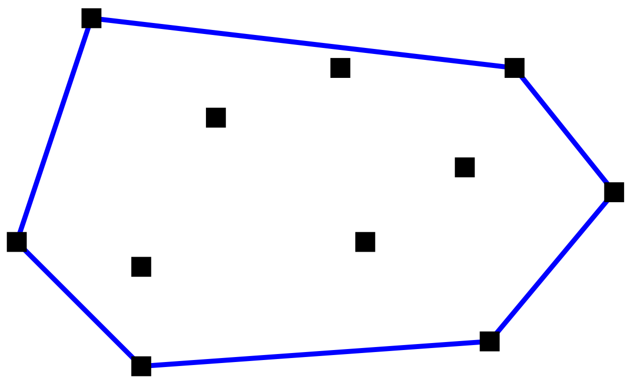
- $2n$ punkti sorteerimine — $O(n \log n)$.
- Tsükkel üle $2n$ punkti, igal iteratsioonil
  - kolm operatsiooni järjendiga T, keerukus $O(\log n)$.
  - kuni kaks lõikude lõikumise kontrolli, keerukus $O(1)$.

  Seega kokku $O(n \log n)$.

Kokku on siis tööaeg $O(n \log n)$.

Kõikvõimalikke lõikumisi pole selle ajaga võimalik üles lugeda, neid võib olla $\Omega(n^2)$.

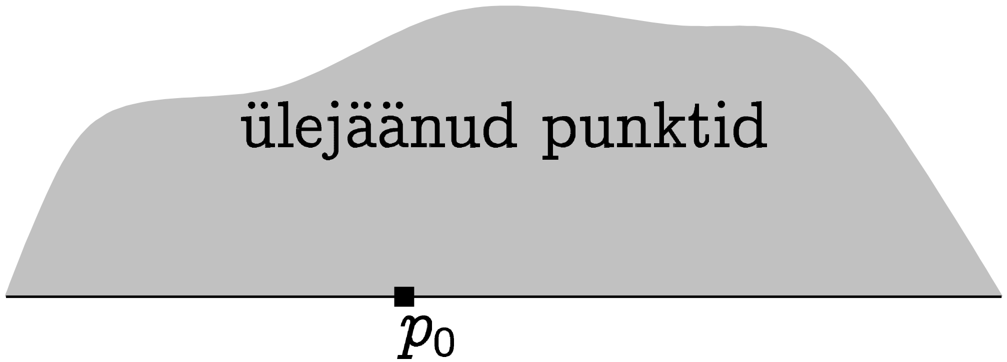
Olgu antud mingid punktid $p_0, p_1, \ldots, p_n$. Nende *kumeraks katteks* nimetatakse vähimat kumerat hulknurka, mis kõiki neid punkte sisaldab.



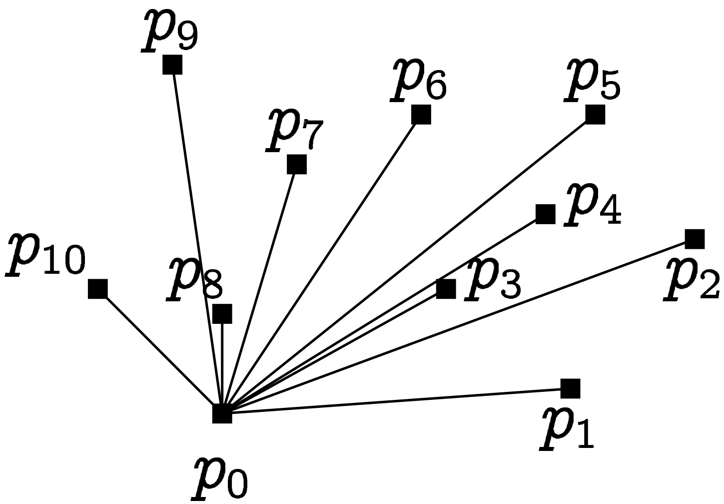
Kumera katte tipud on mingites neist punktidest.

Kuidas leida punktihulga kumerat katet?

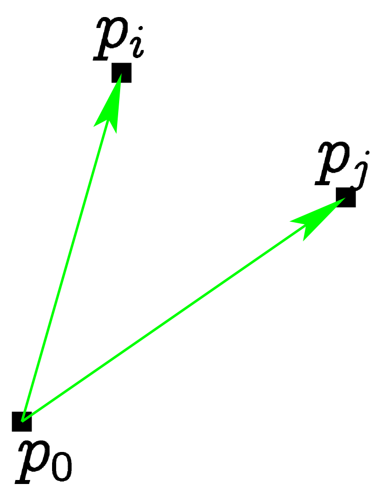
Grahami seiremeetod. Olgu p_0 kõige väiksema y-koordinaadiga punkt. Kui neid on mitu, siis kõige väiksema x-koordinaadiga.



Sorteerime ülejäänud punktid p_i vektorite $\overrightarrow{p_0 p_i}$ tõusu järgi.



Punktide võrdlemiseks vaatame, kumba pidi on järgmised vektorid.



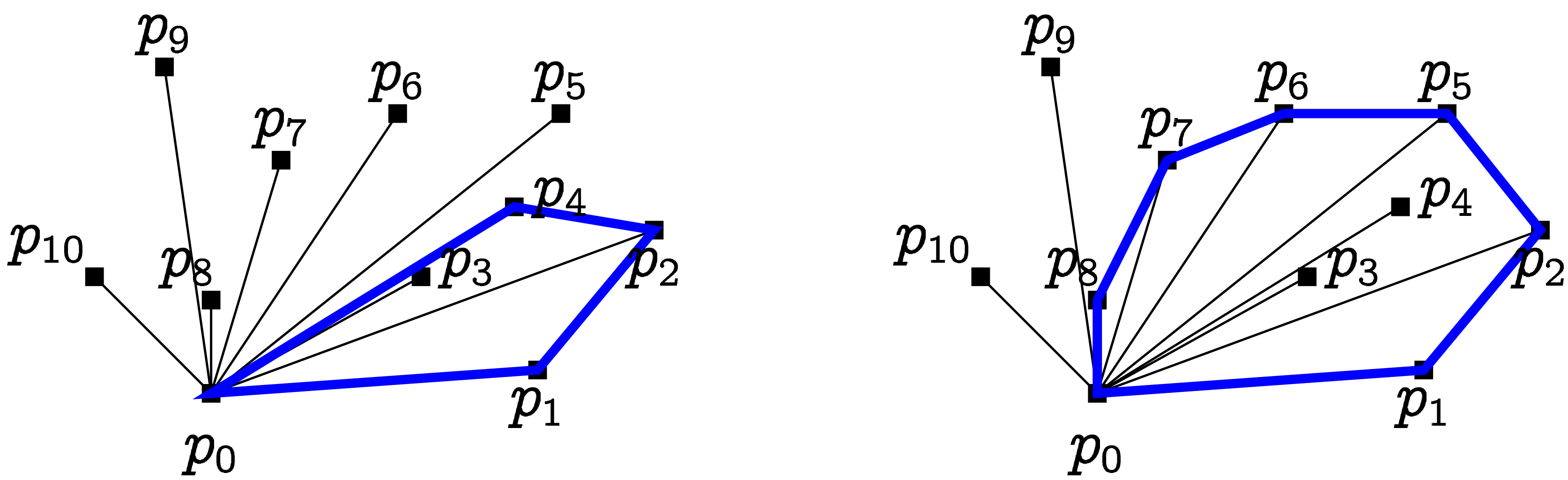
S.t. tõusunurki ei tule välja arvutada.

Kui mitmel punktil on sama tõusunurk, siis võime arvesse võtta ainult kaugeima punkti. Lähemad punktid jäävad kindlasti kumeraks katteks oleva hulknurga sisemusse.

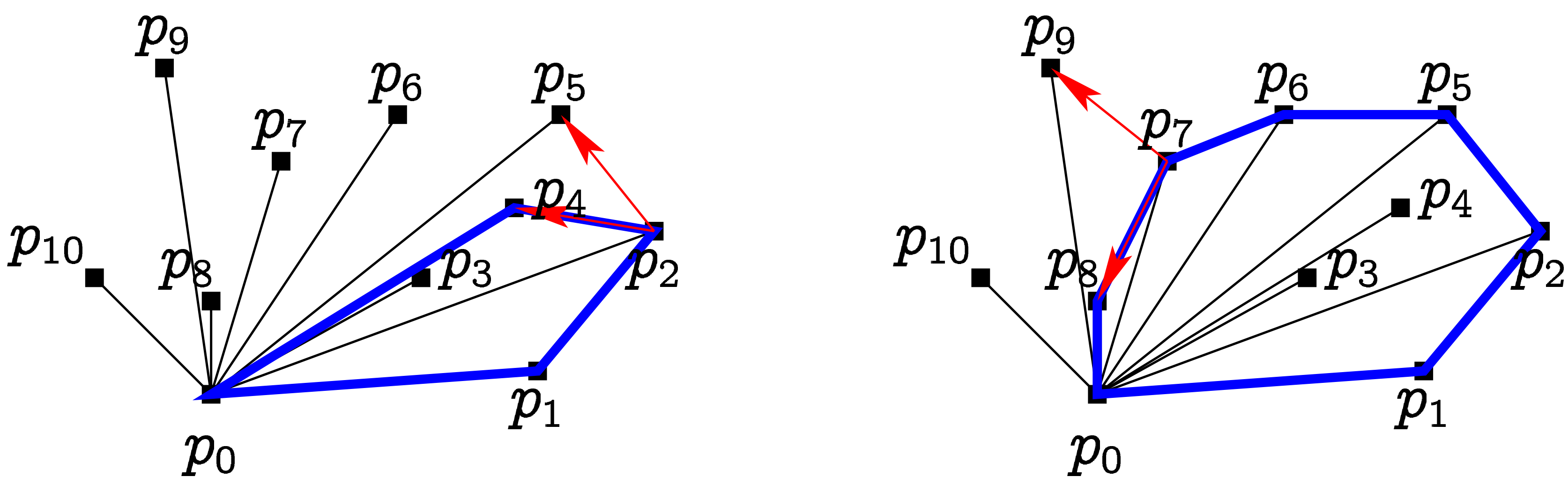
Kaugeim punkt — suurima y-koordinaadiga.

Punktide p_0, p_1, p_2 kumer kate on (p_0, p_1, p_2). (kui p_0, p_1 ja p_2 ei asu ühel sirgel)

Olgu me juba leidnud punktide $p_0, \ldots, p_i$ kumera katte $(q_0, \ldots, q_j)$. Vaatame punkti p_{i+1}. Ta kuulub kindlasti punktide $p_0, \ldots, p_{i+1}$ kumerasse kattesse.

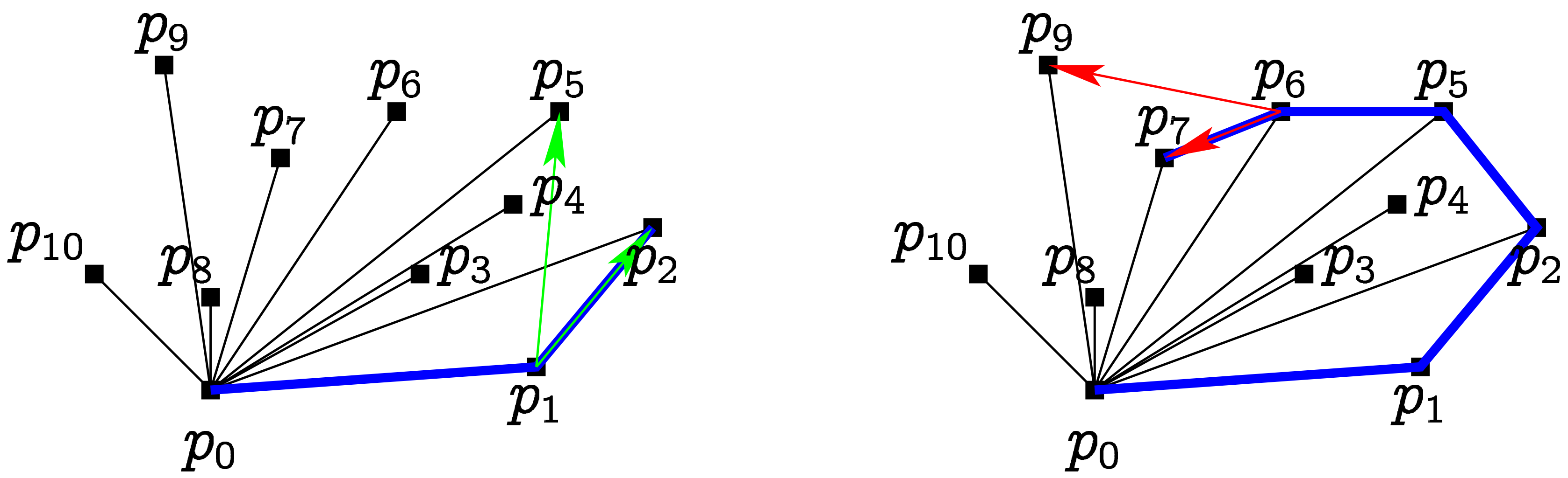


Kontrollimaks, kas q_j kuulub $p_0, \ldots, p_{i+1}$ kumerasse kattesse, vaatame vektoreid $\overrightarrow{q_{j-1}q_j}$ ja $\overrightarrow{q_{j-1}p_{i+1}}$.

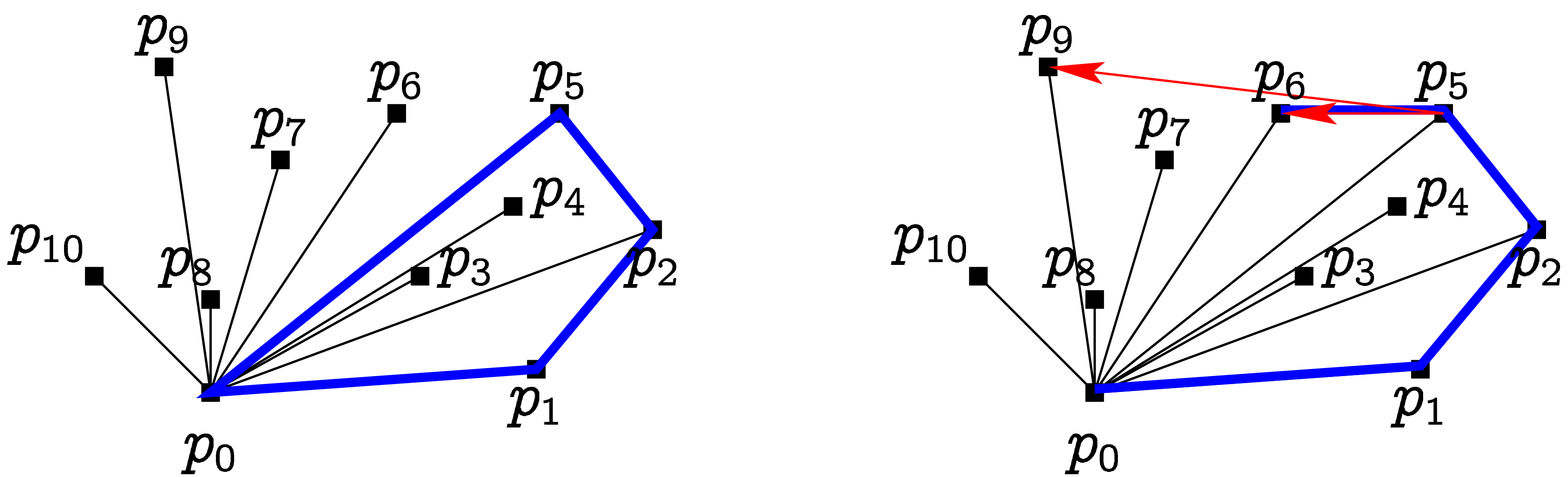


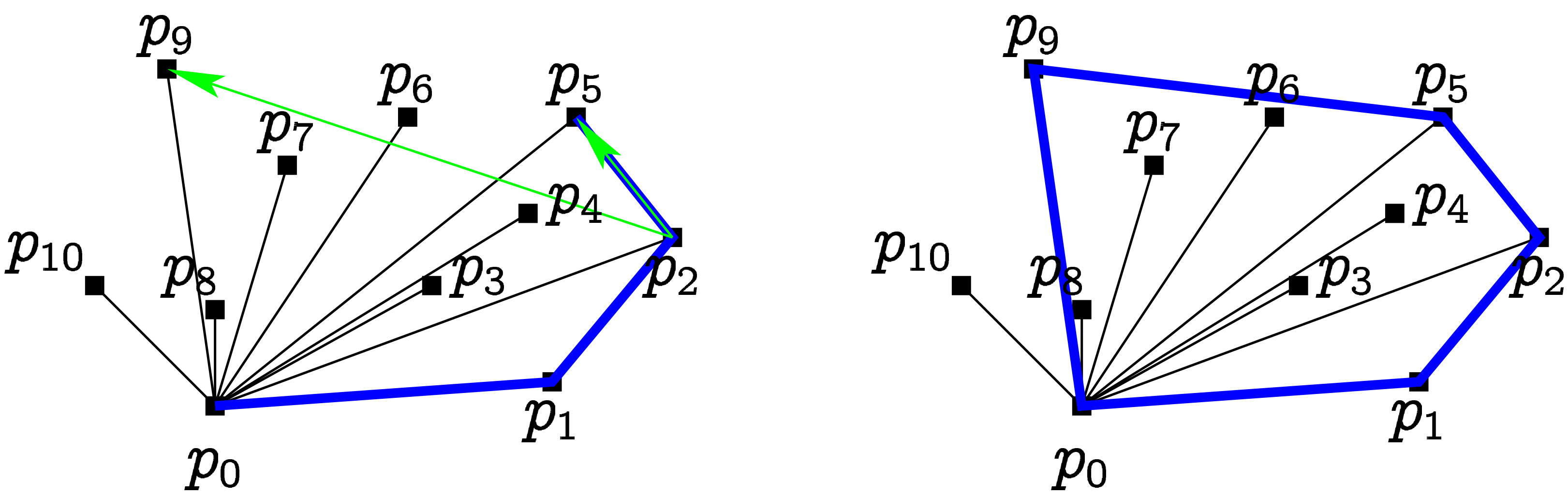
Kui $\overrightarrow{q_{j-1}p_{i+1}}$ asub $\overrightarrow{q_{j-1}q_j}$-st paremal, siis ei kuulu q_j punktide $p_0, \ldots, p_{i+1}$ kumerasse kattesse.

Sel juhul vähendame j-i ühe võrra ja proovime uuesti.



Kui $\overrightarrow{q_{j-1}p_{i+1}}$ asub $\overrightarrow{q_{j-1}q_j}$-st vasakul, siis oleme $p_0, \ldots, p_{i+1}$ kumera katte leidnud. See on $(q_0, \ldots, q_j, p_{i+1})$.





Algoritmi realiseerides on mõttekas punkte $q_0, \ldots, q_j$ pinus hoida, nii et q_j on pealmine.

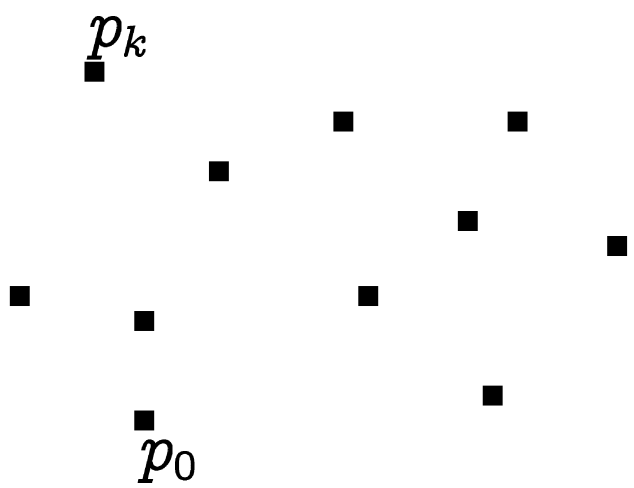
Keerukus:

- Vähima y-koordinaadiga punkti otsimine — $O(n)$.
- Sorteerimine — $O(n \log n)$.
- Kumera katte leidmisel võib iga punkt
  - (täpselt) ühel korral kattesse sattuda;
  - ülimalt ühel korral kattest eemaldatud saada.

  Seega võtab punktide lisamine/eemaldamine aega $O(n)$.
- Igale vektorite omavahelise asendi võrdlemisele järgneb kas mingi punkti lisamine kattesse või eemaldamine sealt. Seega on võrdlusi $O(n)$.

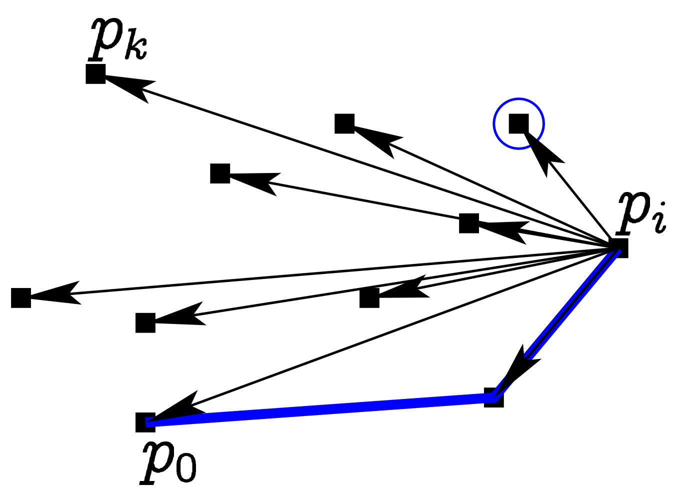
Kokku seega $O(n \log n)$.

Jarvise mähkimismeetod. Olgu p_0 kõige väiksema ja p_k kõige suurema y-koordinaadiga punkt. (kui neid on mitu, siis kõige vasakpoolsem(ad))



Nii p_0 kui ka p_k kuuluvad kumerasse kattesse. Neist kahest punktist „paremale“ jääva kumera katte osa leidmiseks:

Esimeseks punktiks kumera katte parempoolsel osal võtame p_0-i. Kui p_i on viimane leitud punkt kumera katte parempoolsel osal, siis vaatame vektoreid p_i-st kõigisse teistesse punktidesse.



Järgmiseks punktiks võtame parempoolseima vektori teise otstipu. Jätkame, kuni jõuame p_k-ni.

Samamoodi leiame kumera katte vasakpoolse osa.

Keerukus: Punkti lisamisi kumerasse kattesse on samapalju kui kumeras kattes punkte on.

Igal lisamisel tuleb leida parem-/vasakpoolseim punkt. Leidmine on analoogiline massiivi minimaalse elemendi leidmisega. Ühe punkti lisamise keerukus on seega $O(n)$.

Kogukeerukus on $O(nh)$, kus h on tippude arv kumeras kattes.

Valides Grahami ja Jarvise meetodite vahel tuleks võrrelda suurusi h ja $\log n$ (s.t. nende eeldatavate väärtuste proportsioone konkreetses rakenduses).