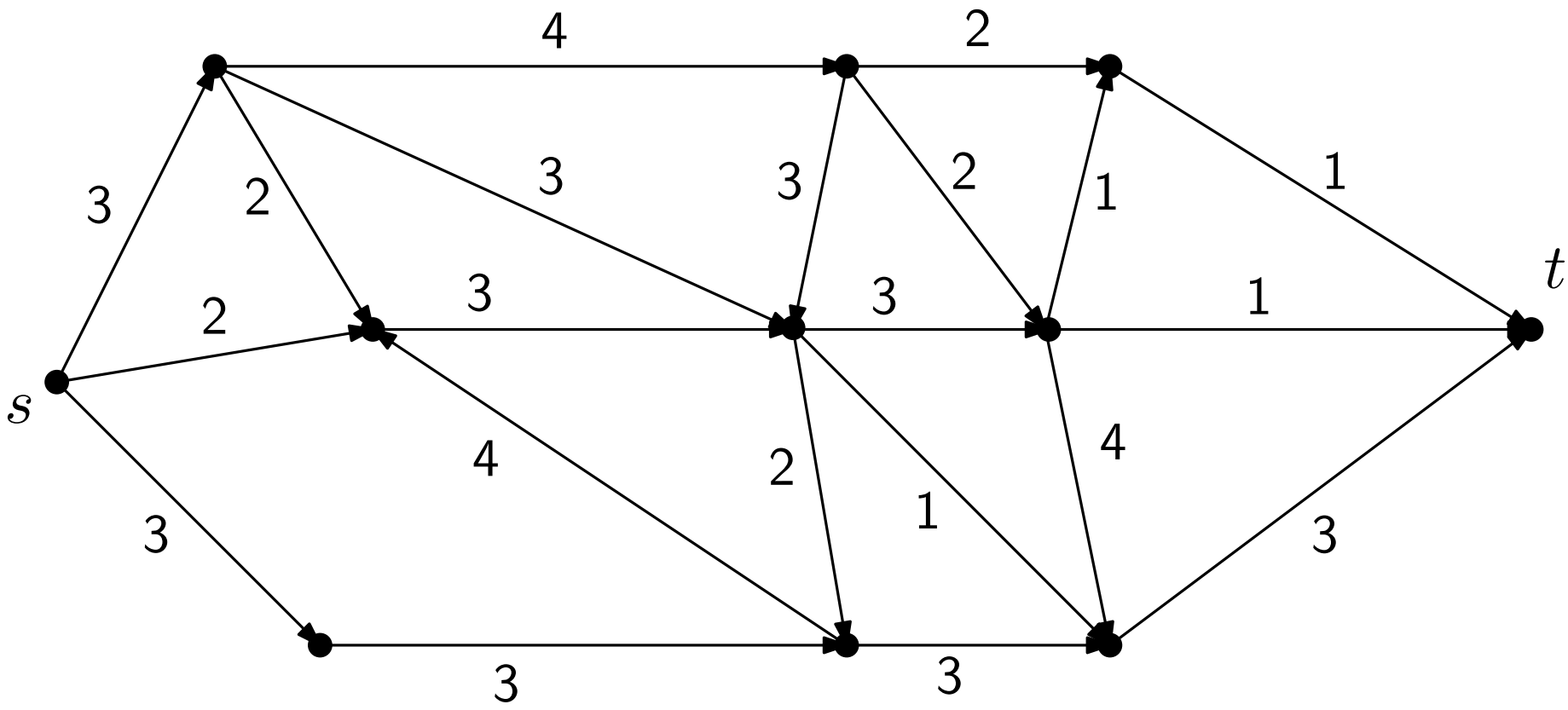
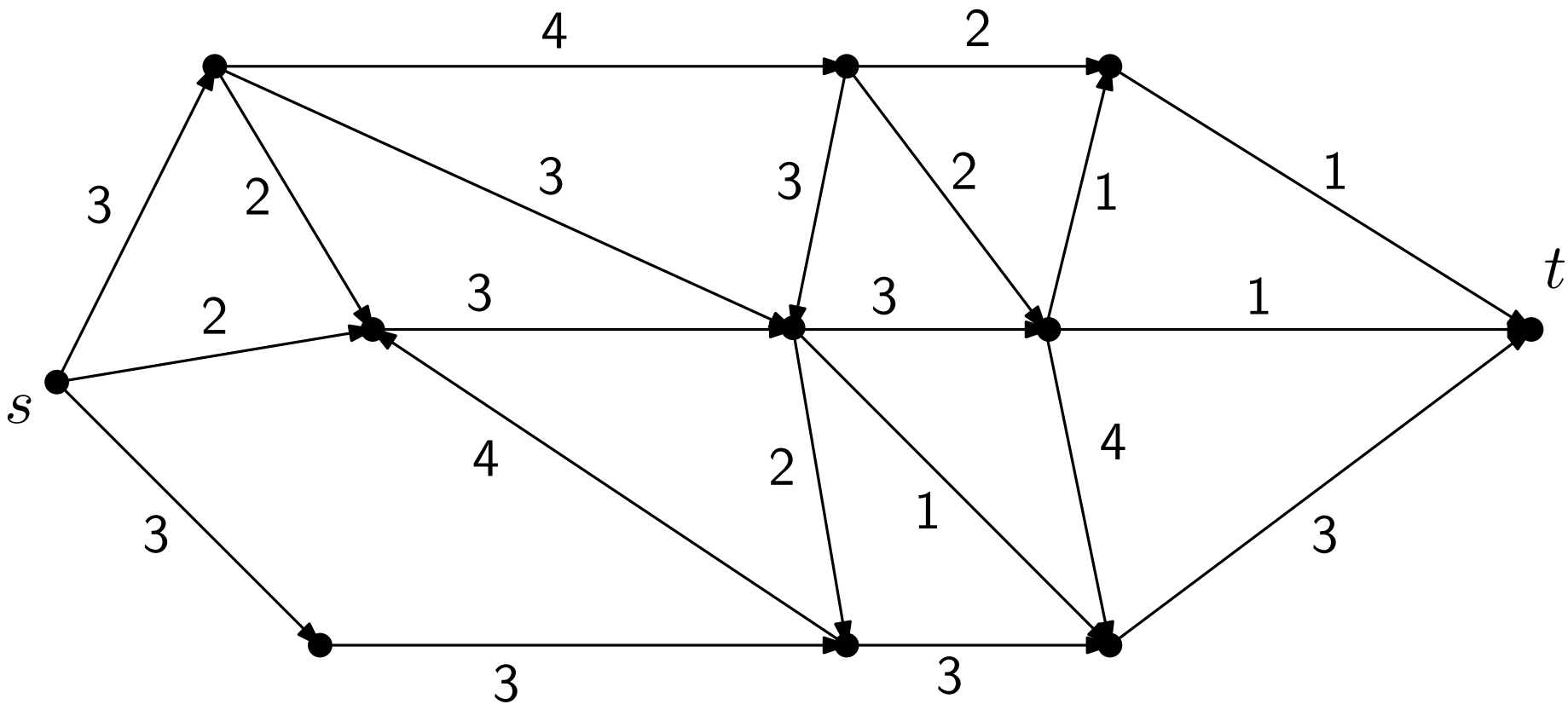
Dinic' Max Flow Algorithm

Slides by Dominik Scheder

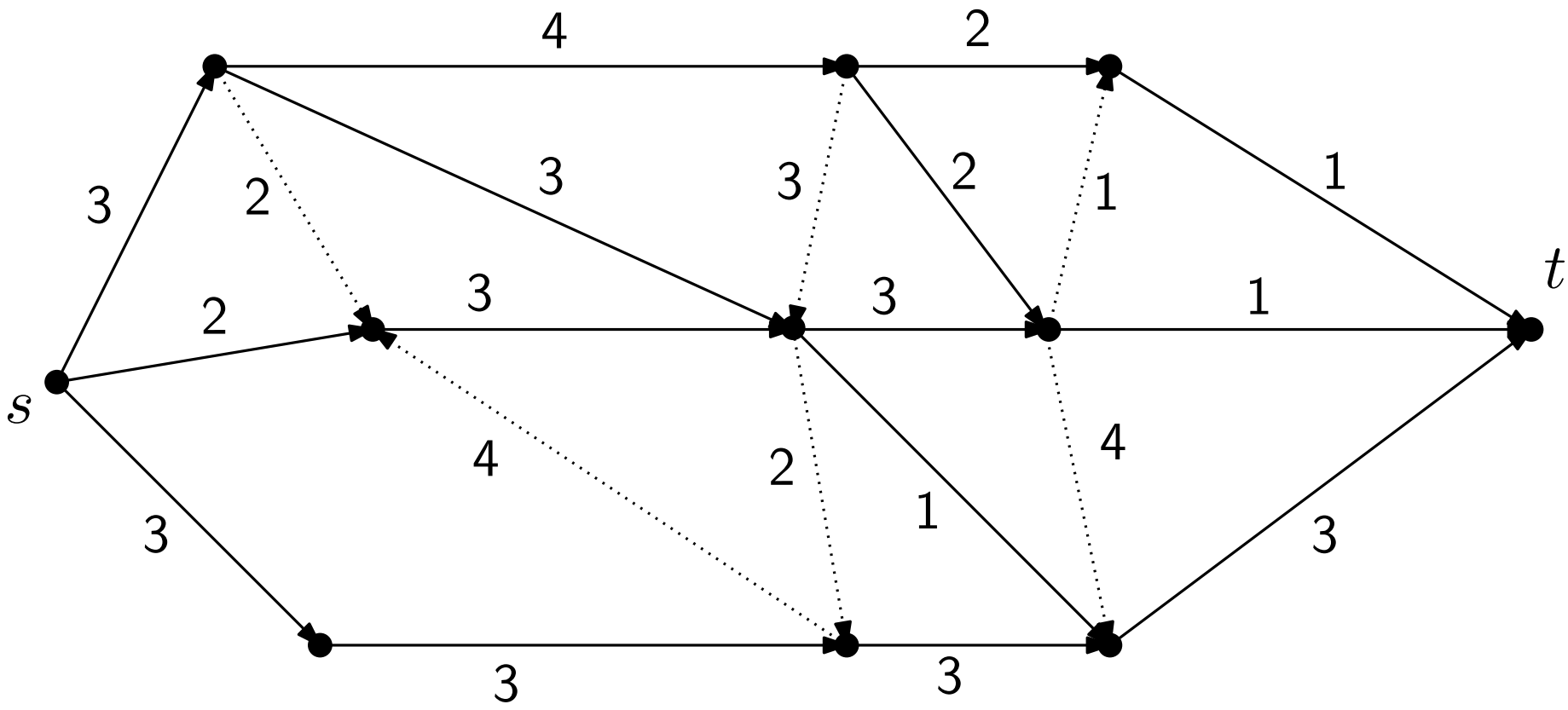
Part I

Dinic' Algorithm in General Flow Networks

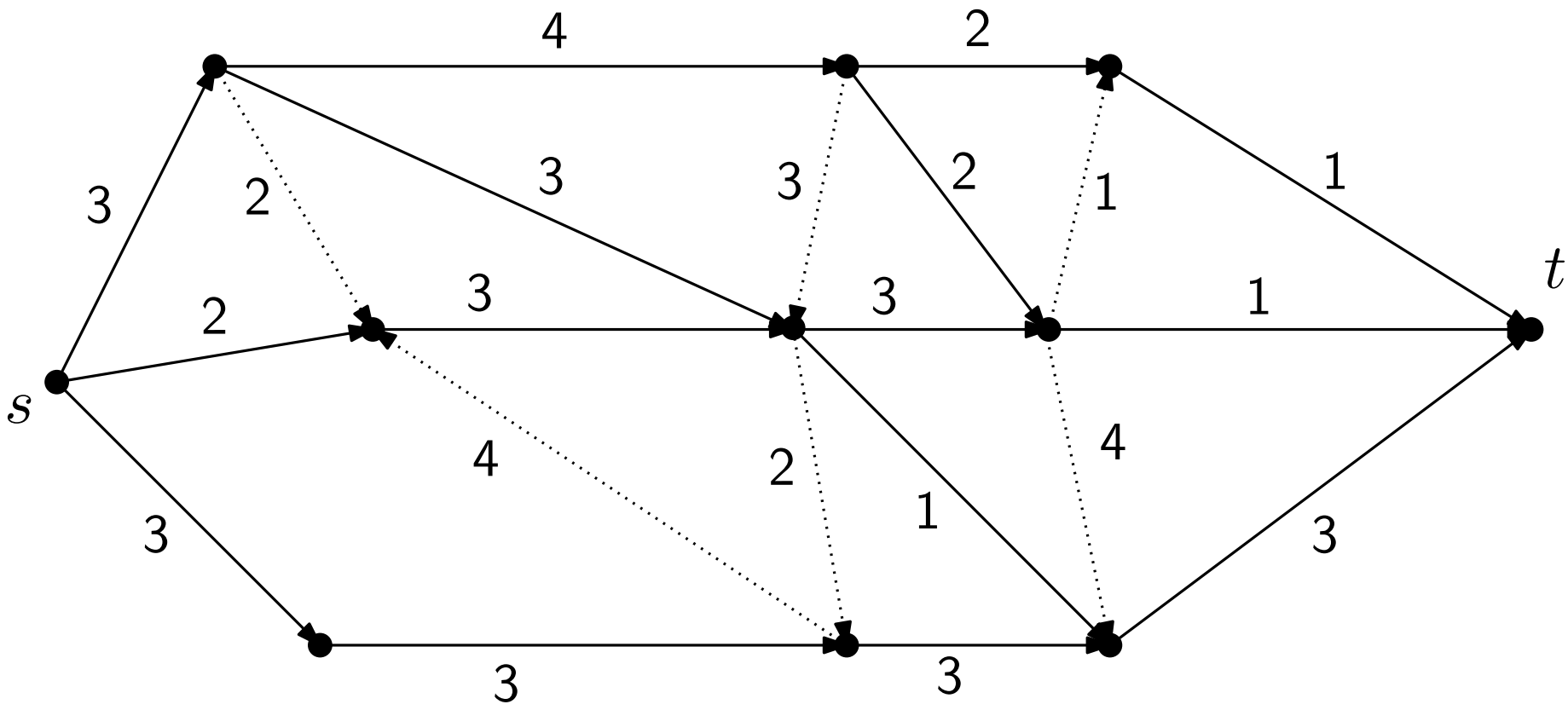

4
2
3
2
3
3
2
1
1
3
2
3
3
1
t
s
4
2
1
4
3
3
3



Find out which edges lie on a shortest s-t-path,

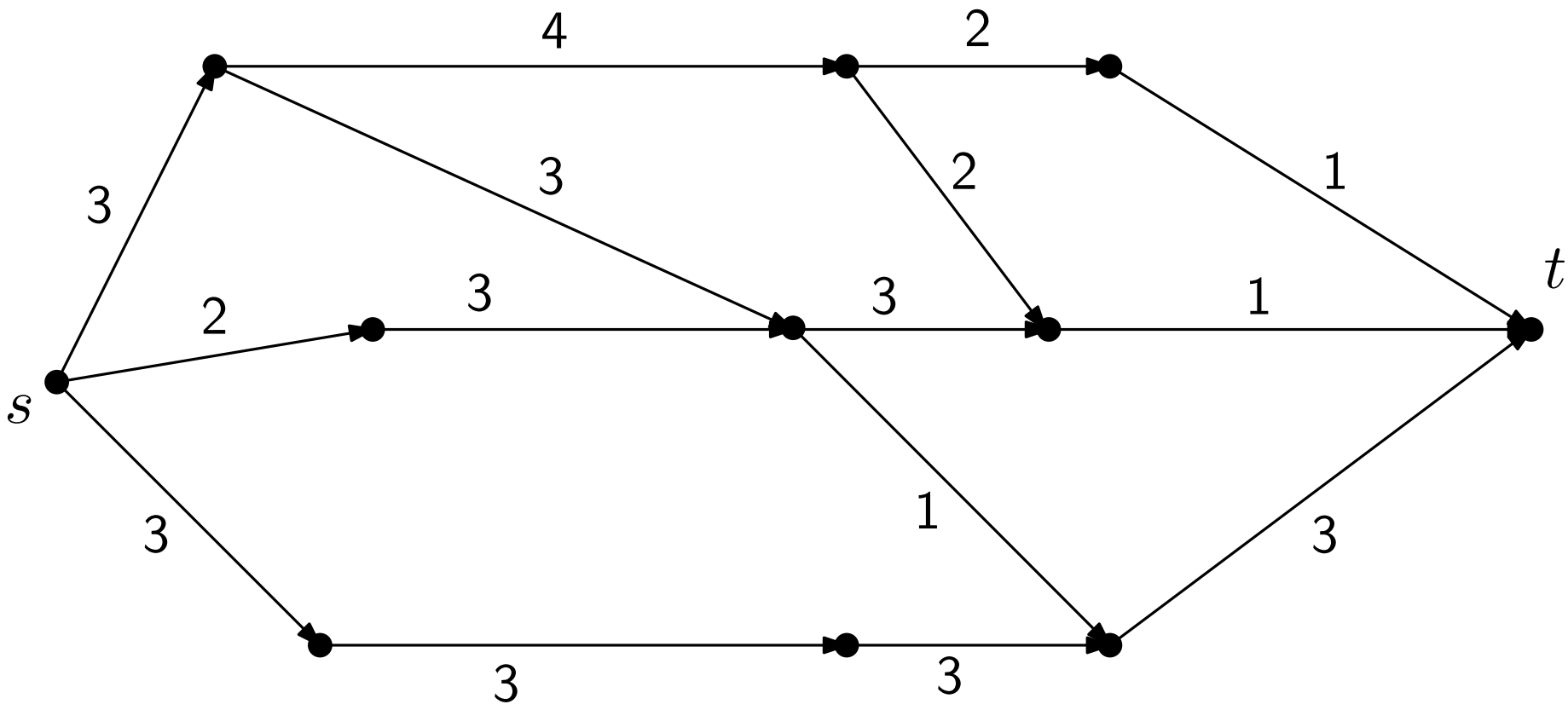


Find out which edges lie on a shortest s-t-path,



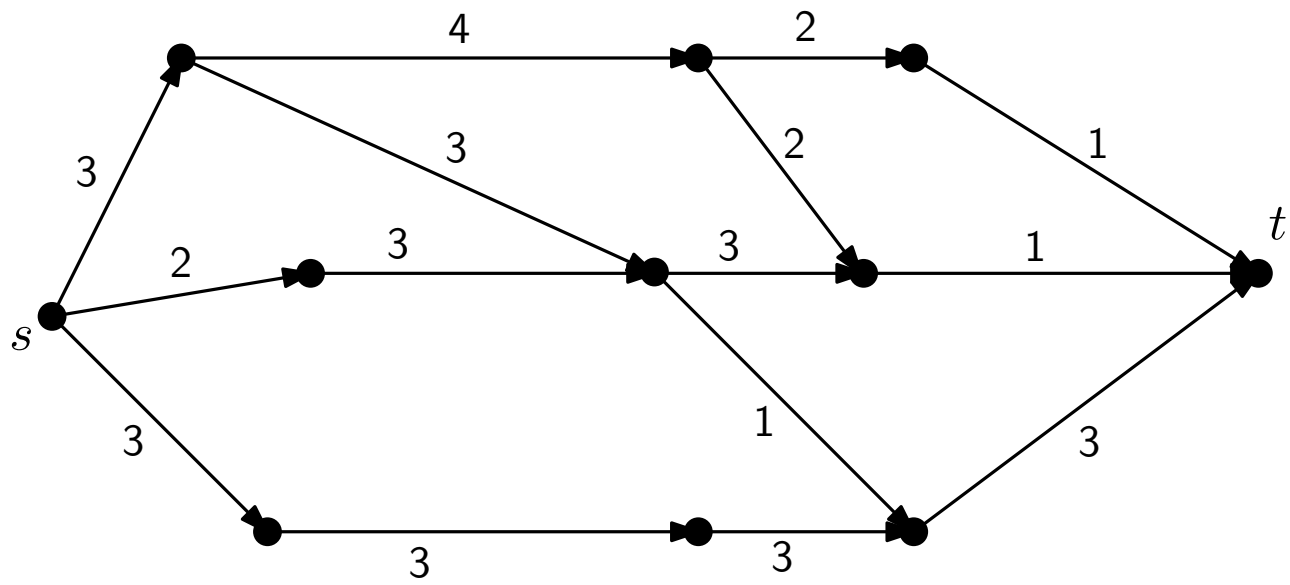
Find out which edges lie on a shortest s-t-path,

and forget the rest for the time being.

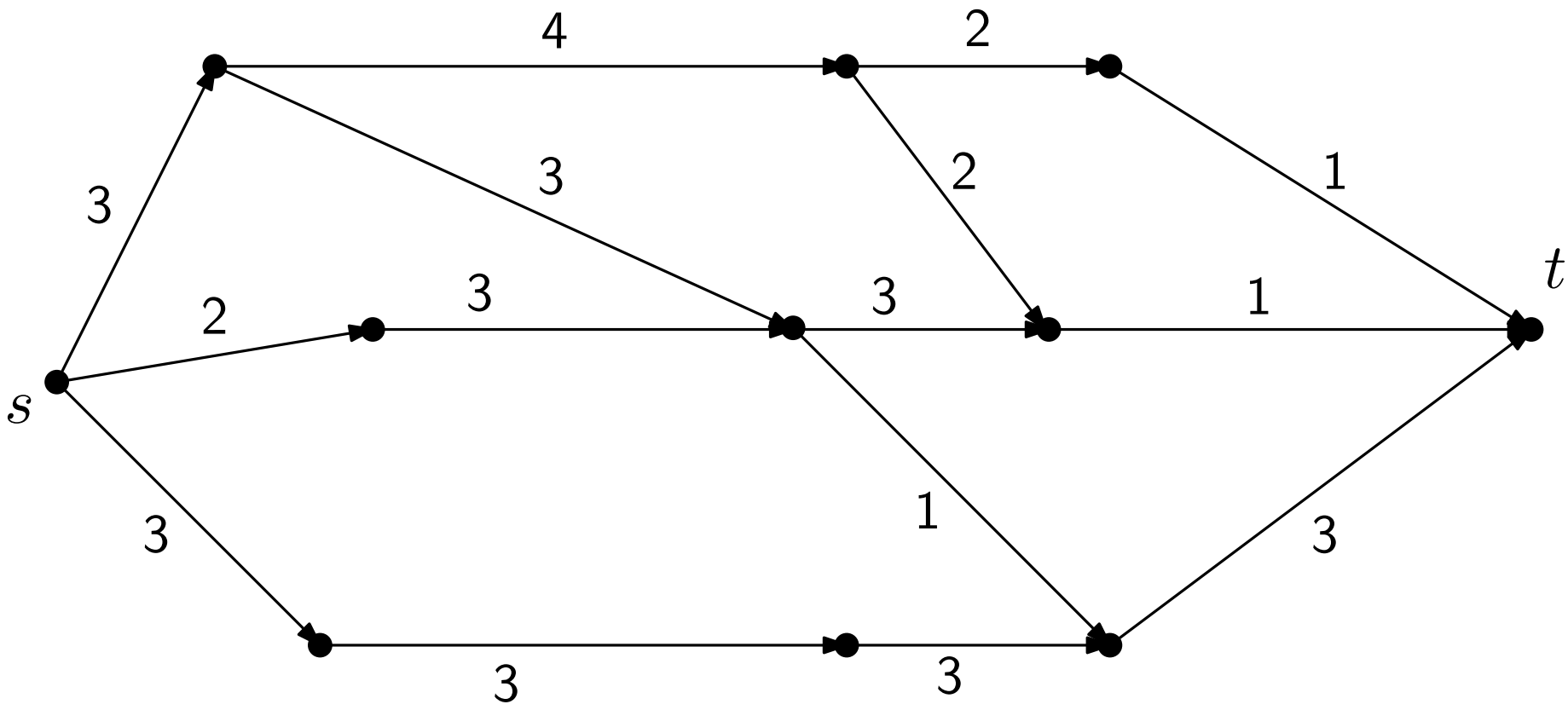


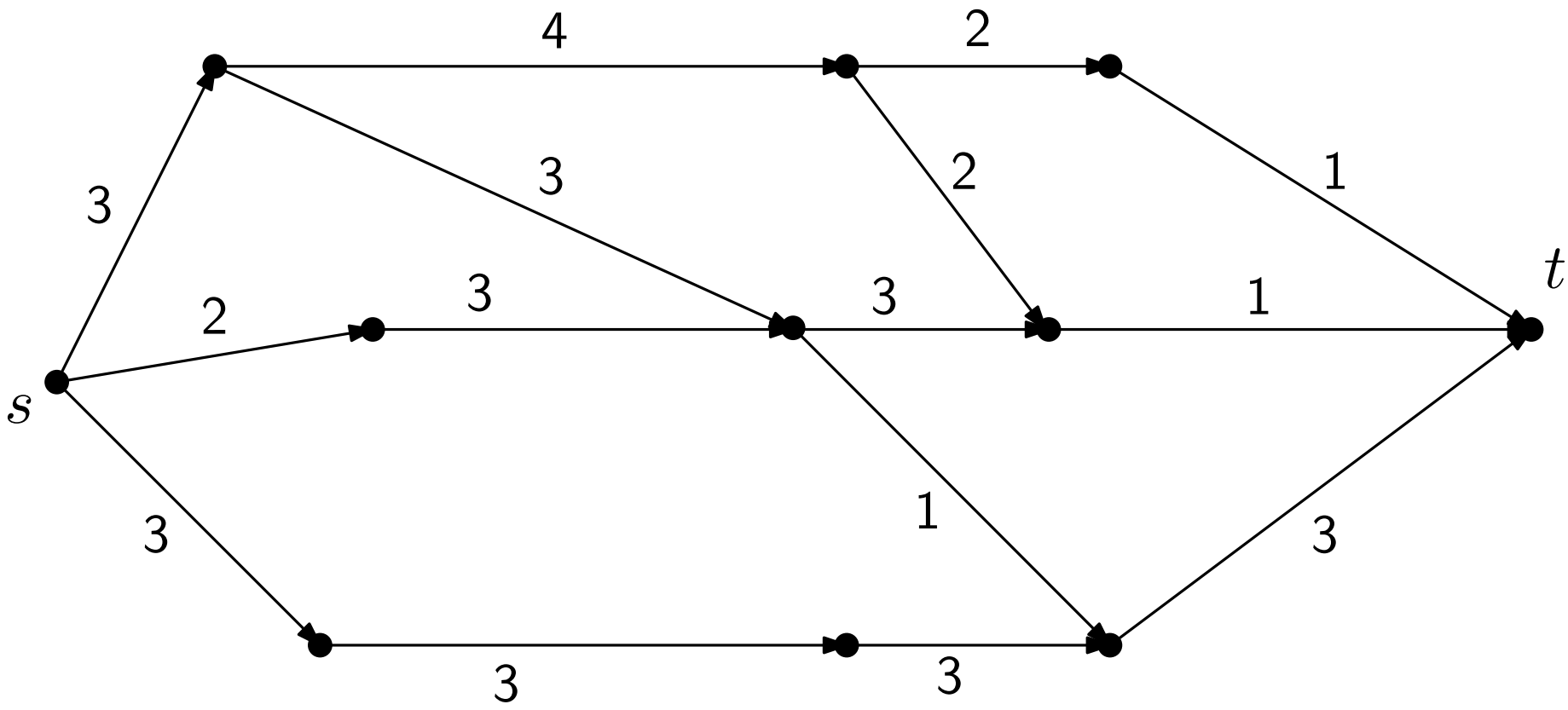
Find out which edges lie on a shortest s-t-path,

and forget the rest for the time being.

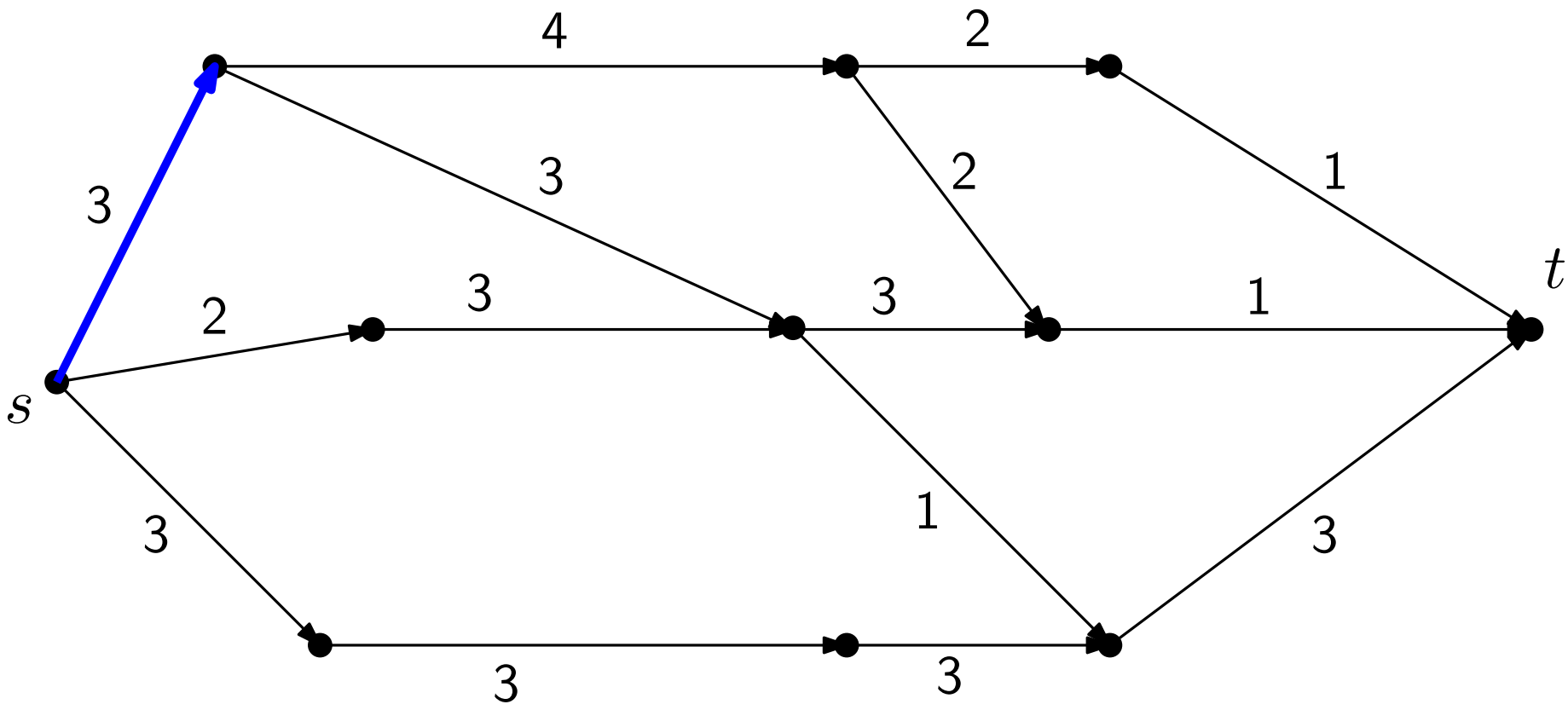


Now lets greedily route as much flow as possible in this network!

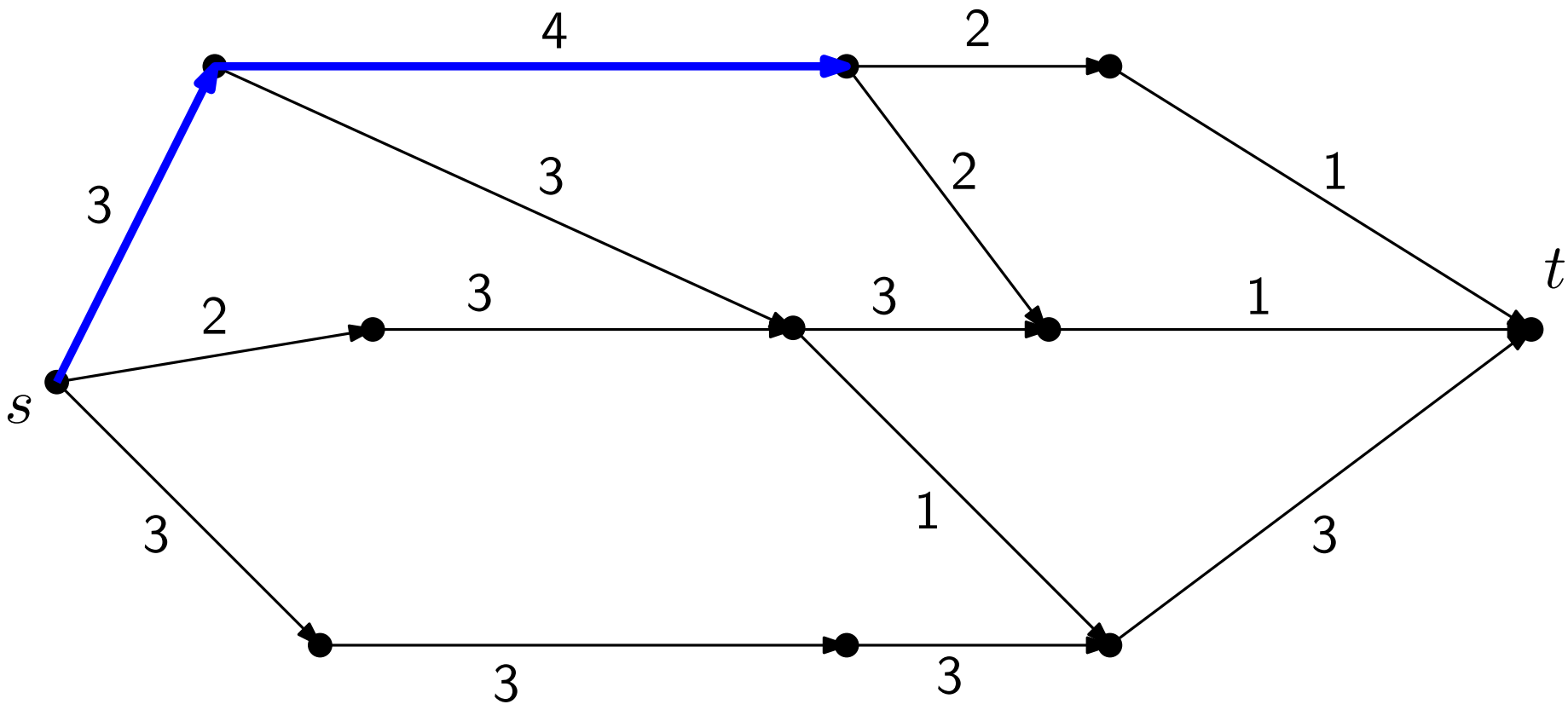

4
2
3
3
2
1
t
3
2
3
3
1
s
1
3
3
3



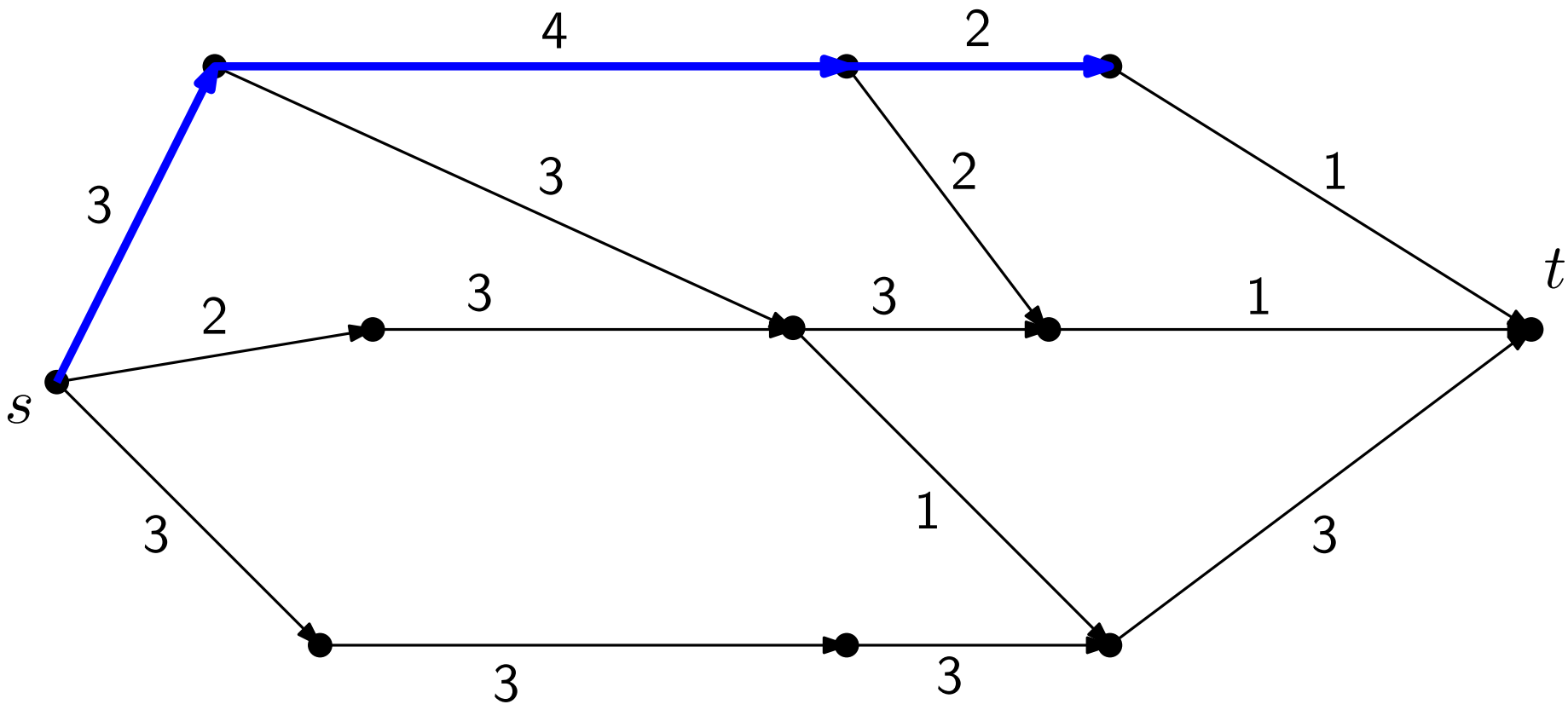
Find an s-t-path with depth-first search.



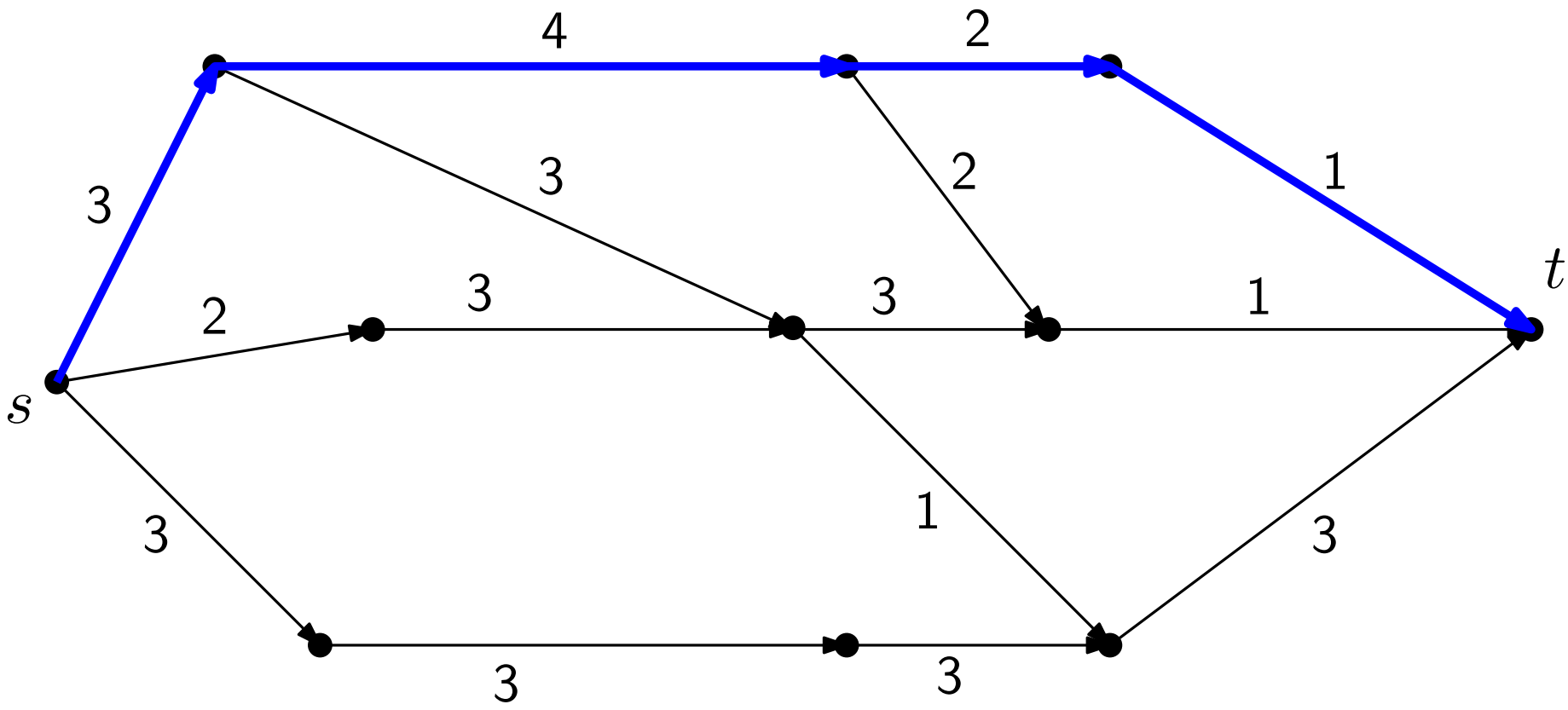
Find an s-t-path with depth-first search.



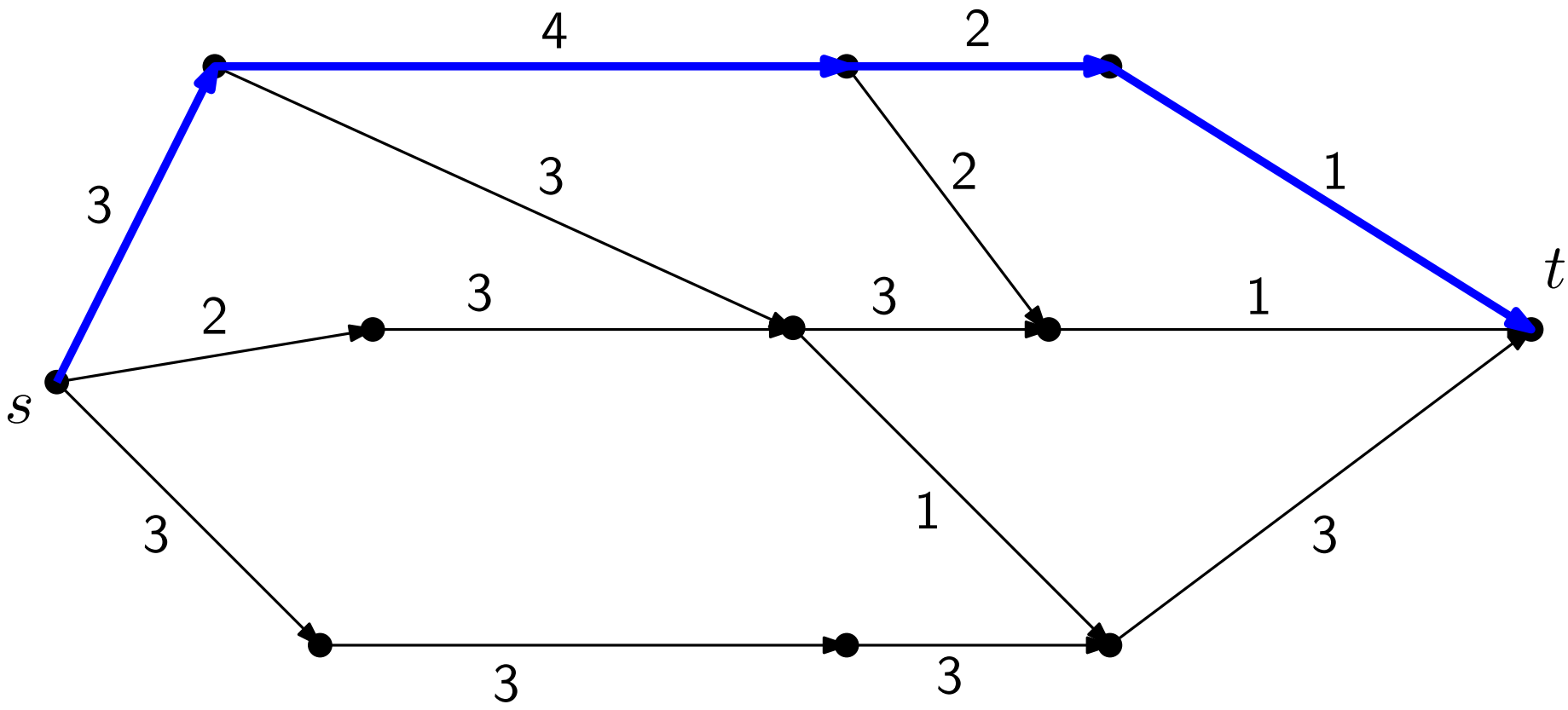
Find an s-t-path with depth-first search.



Find an s-t-path with depth-first search.

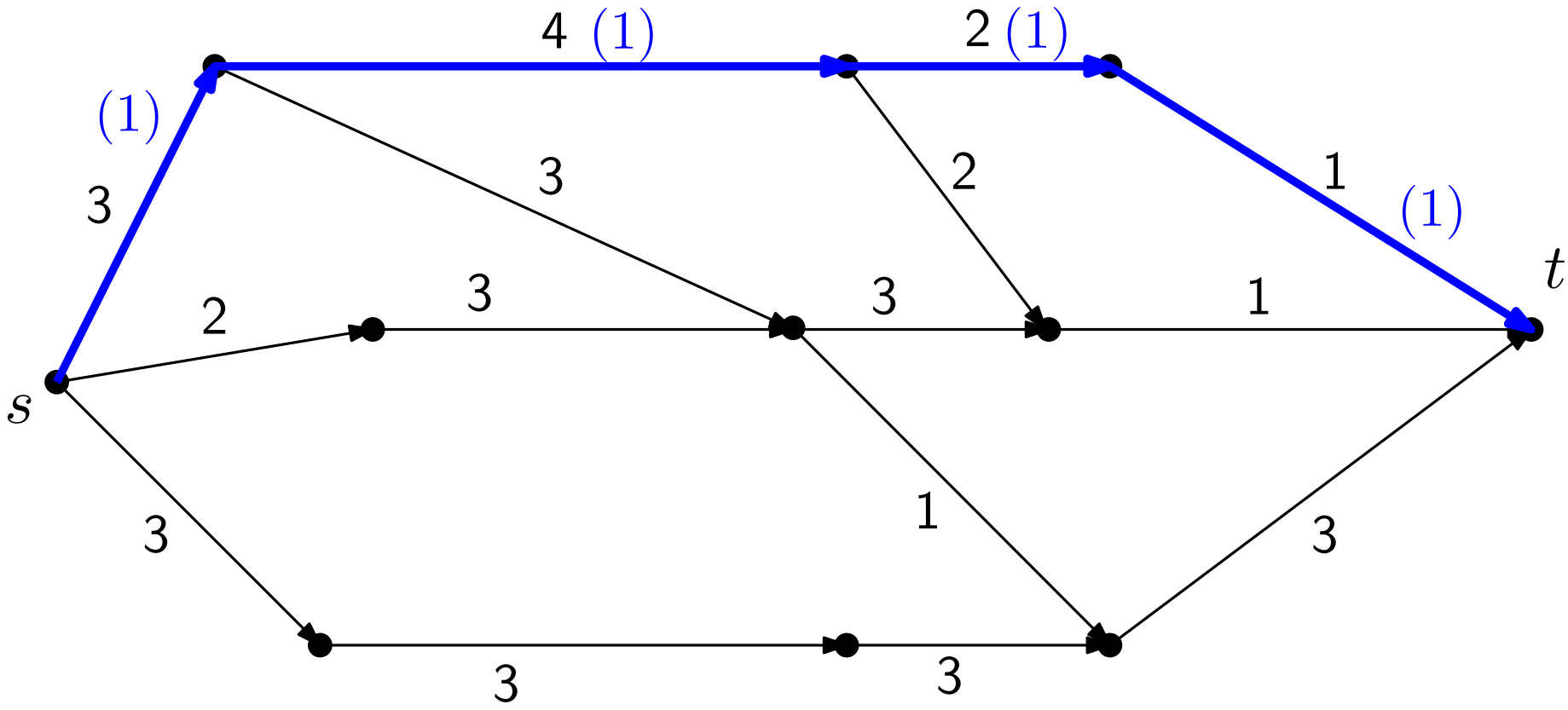


Find an s-t-path with depth-first search.



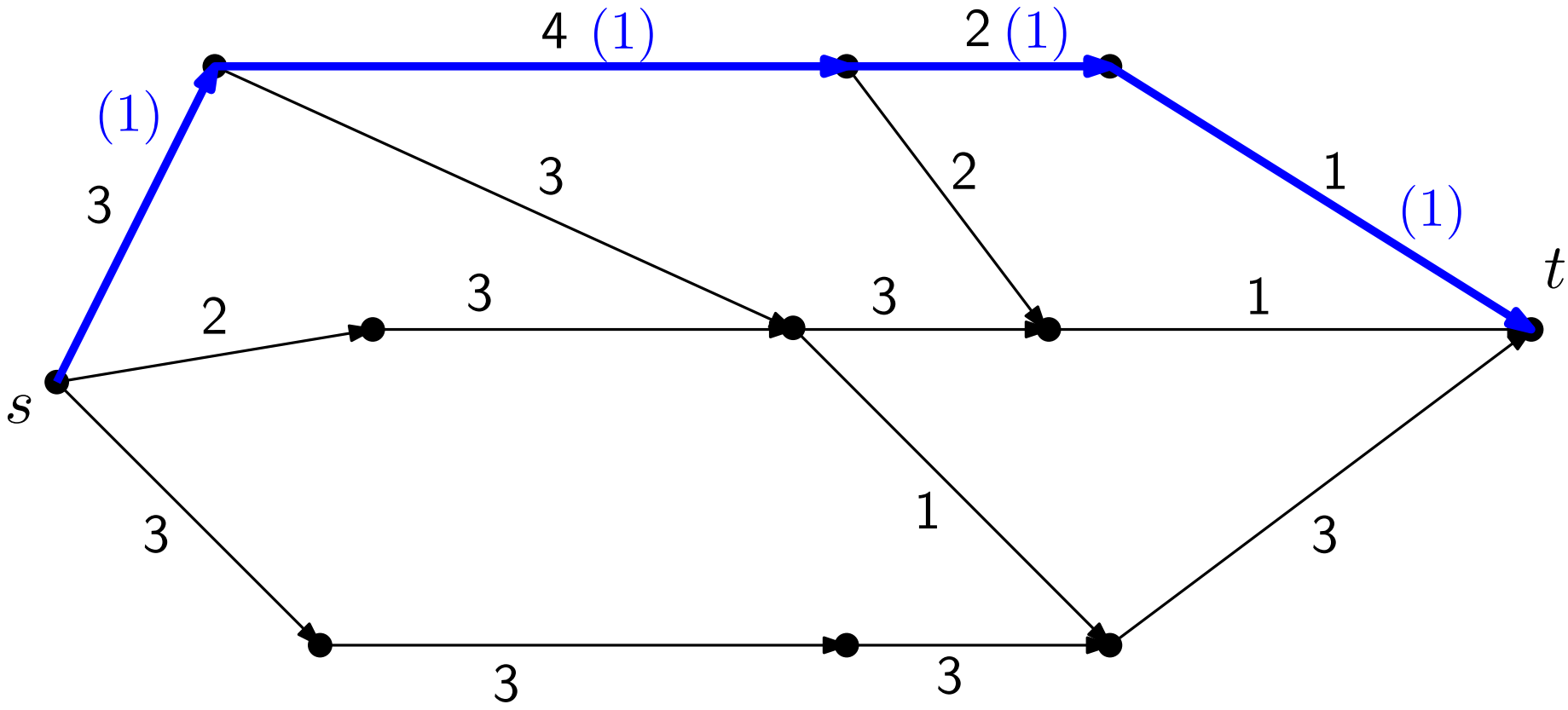
Find an s-t-path with depth-first search.

Route as much flow through it as possible: 1 unit.



Find an s-t-path with depth-first search.

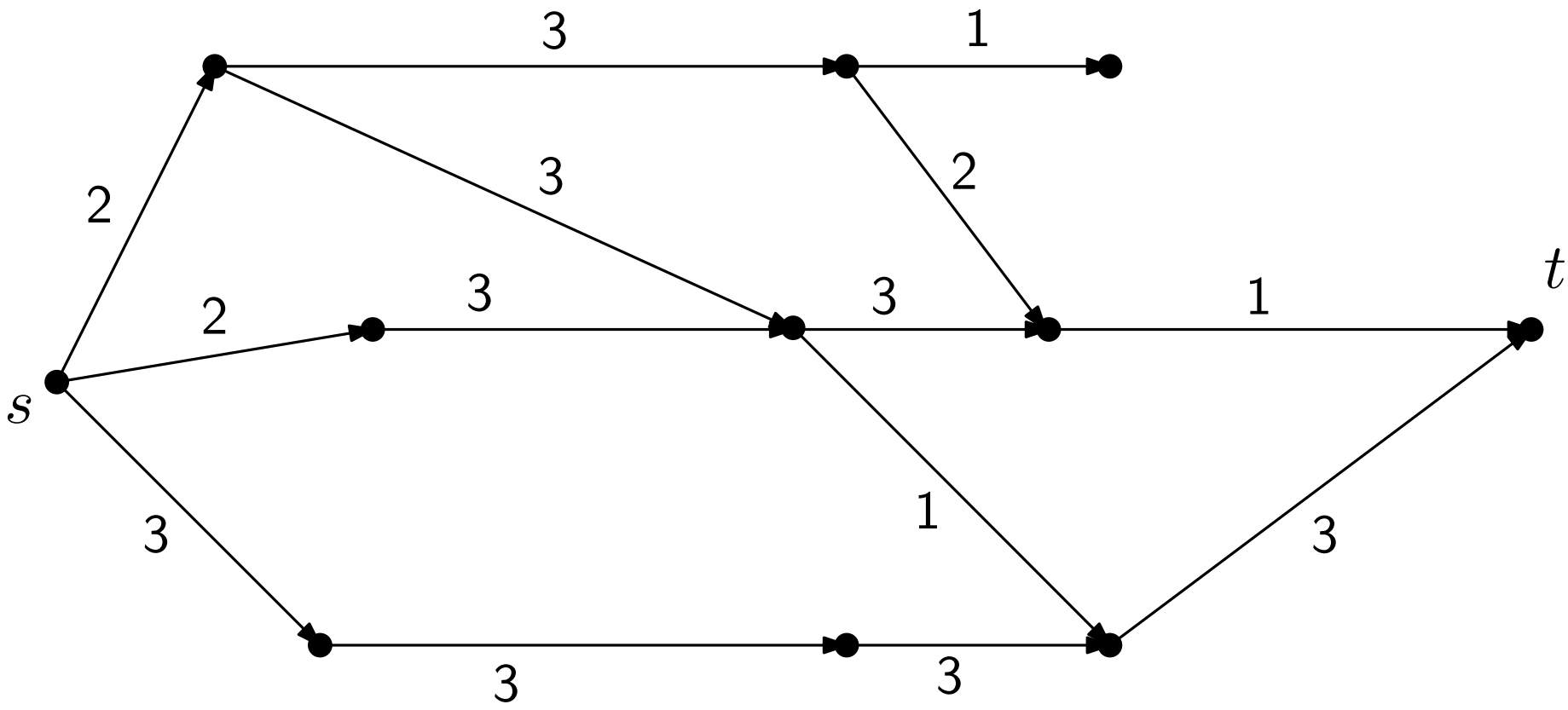
Route as much flow through it as possible: 1 unit.



Find an s-t-path with depth-first search.

Route as much flow through it as possible: 1 unit.

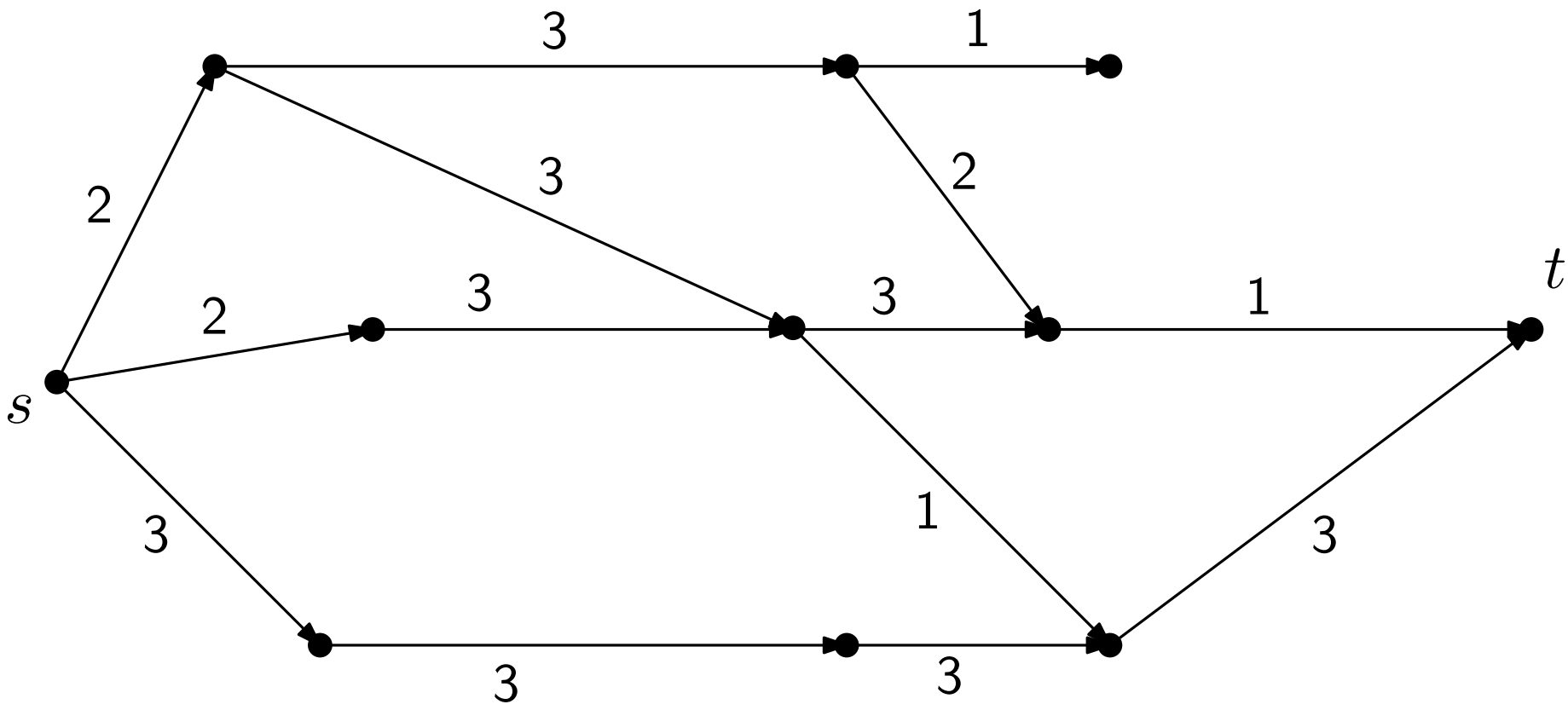
Update capacities.



Find an s-t-path with depth-first search.

Route as much flow through it as possible: 1 unit.

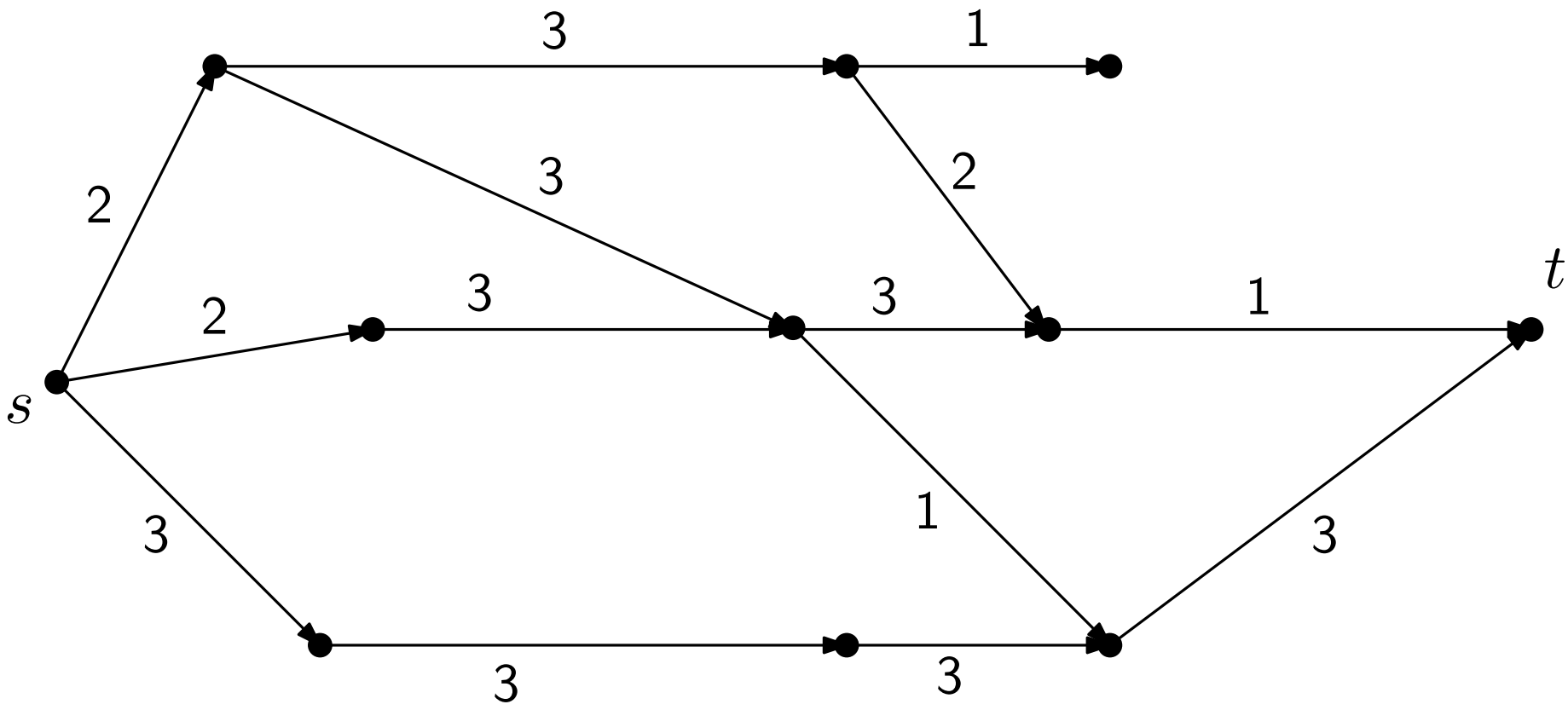
Update capacities.



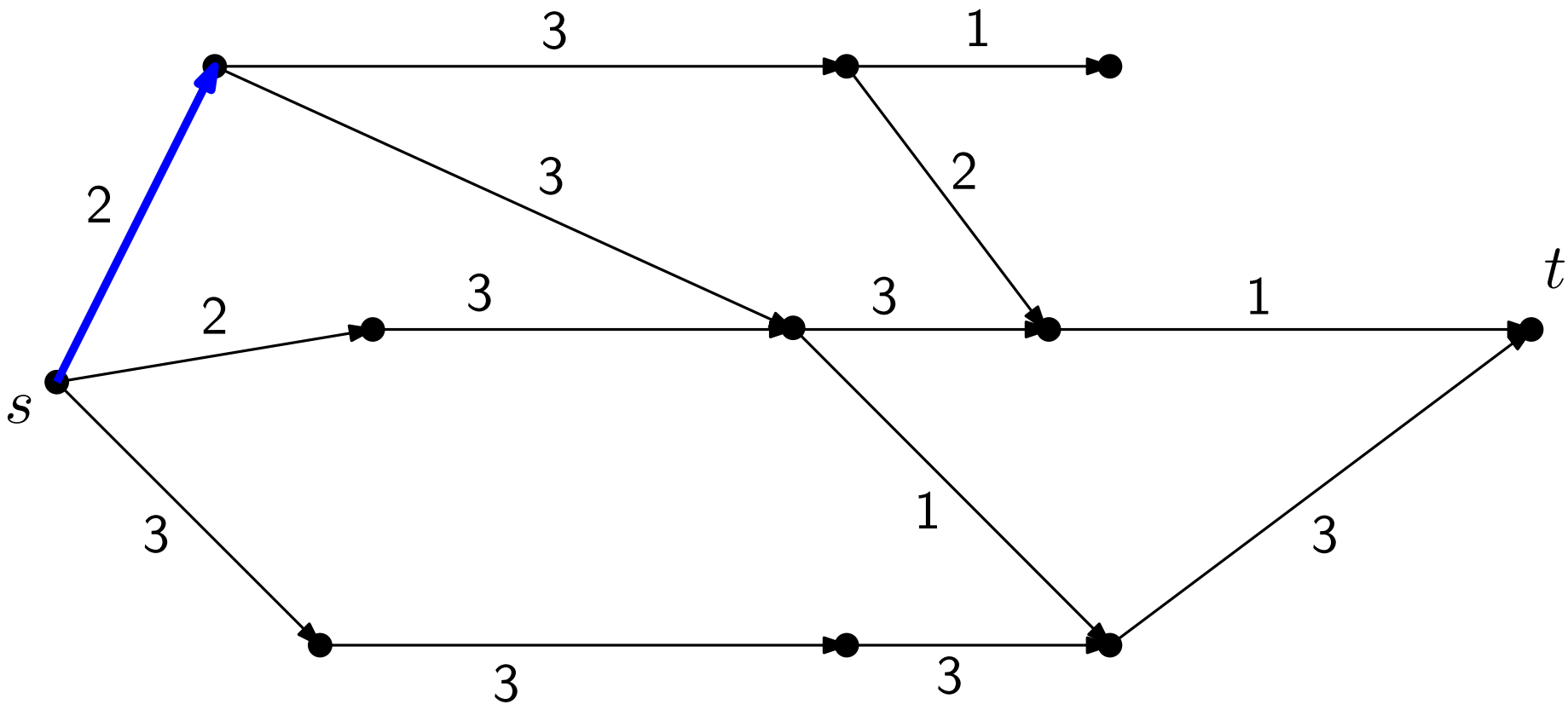
Find an s-t-path with depth-first search.

Route as much flow through it as possible: 1 unit.

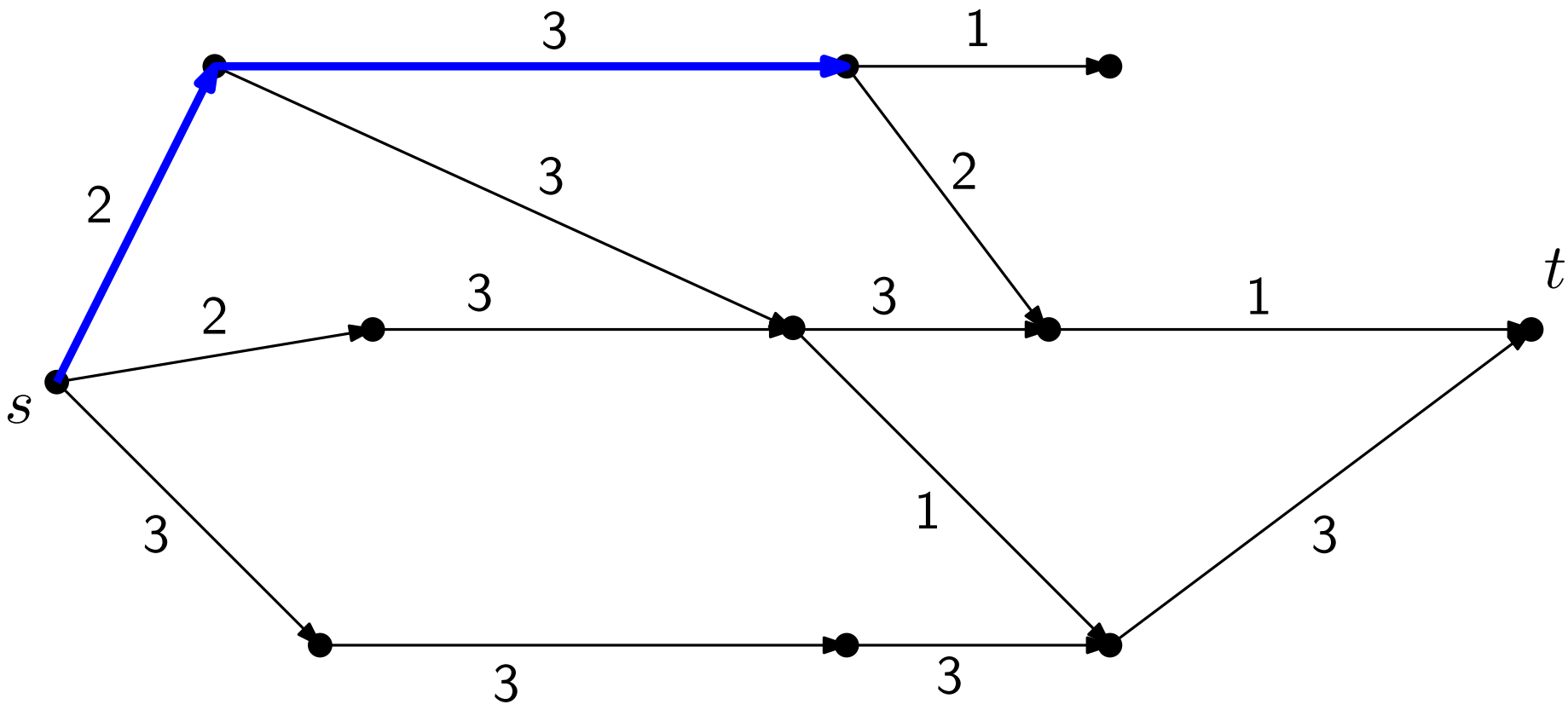
Update capacities. Repeat!



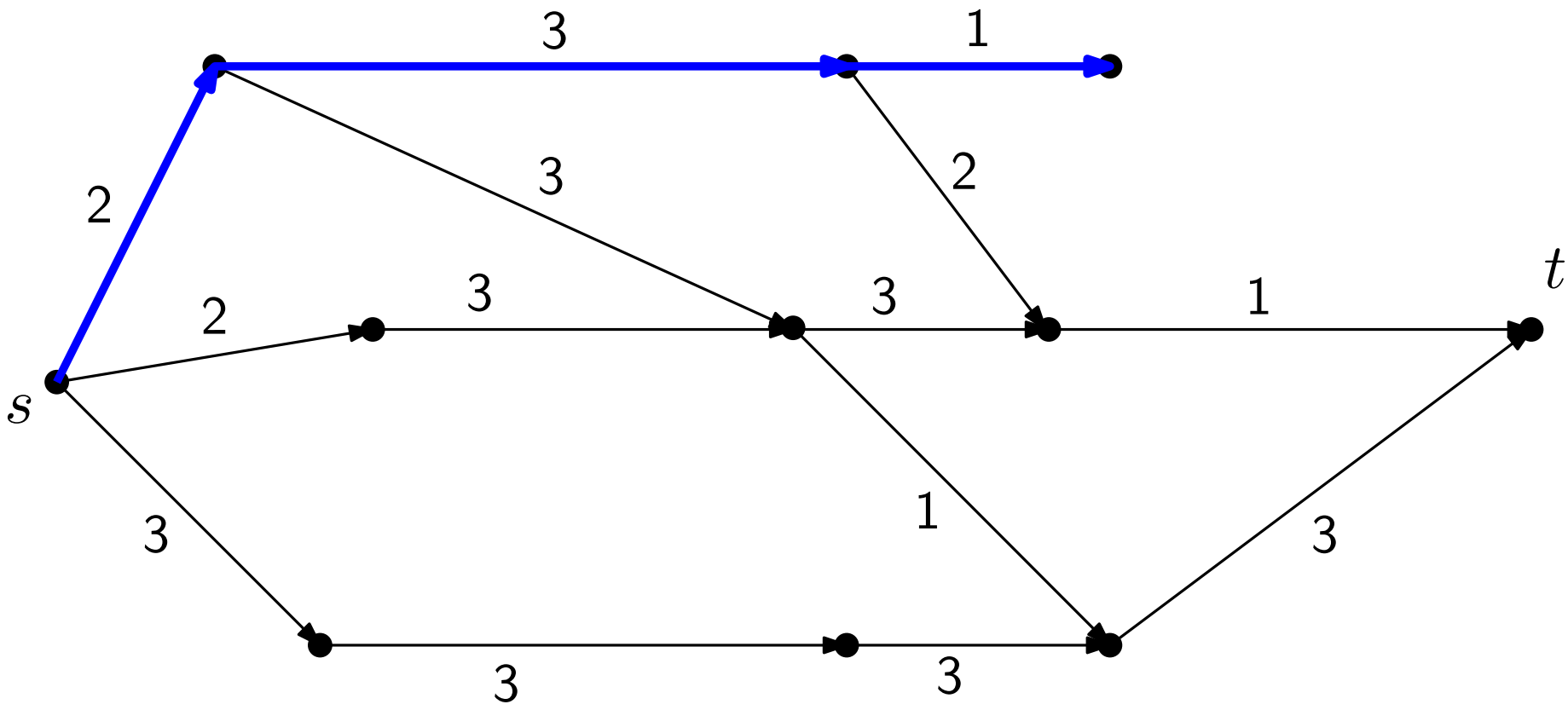
Find an s-t-path with depth-first search.



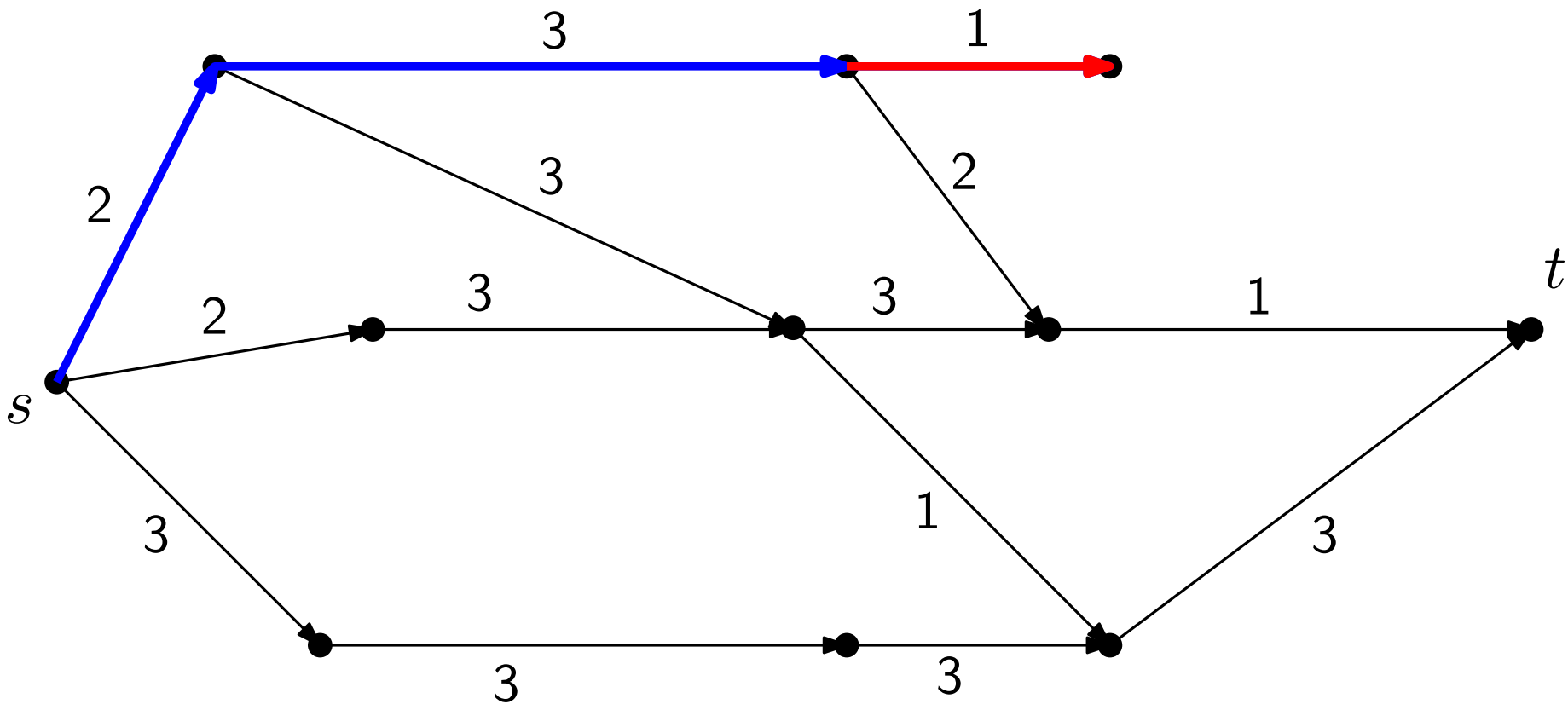
Find an s-t-path with depth-first search.



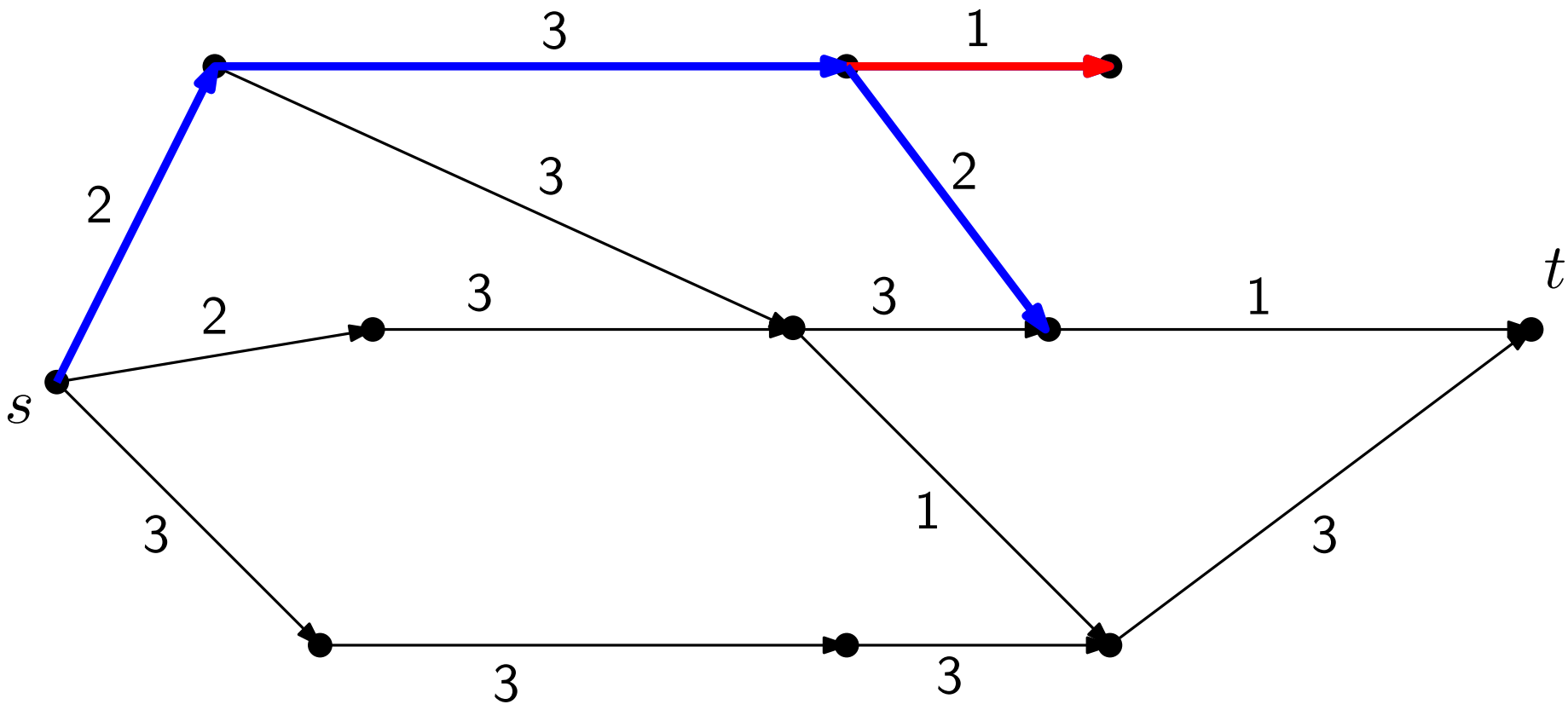
Find an s-t-path with depth-first search.



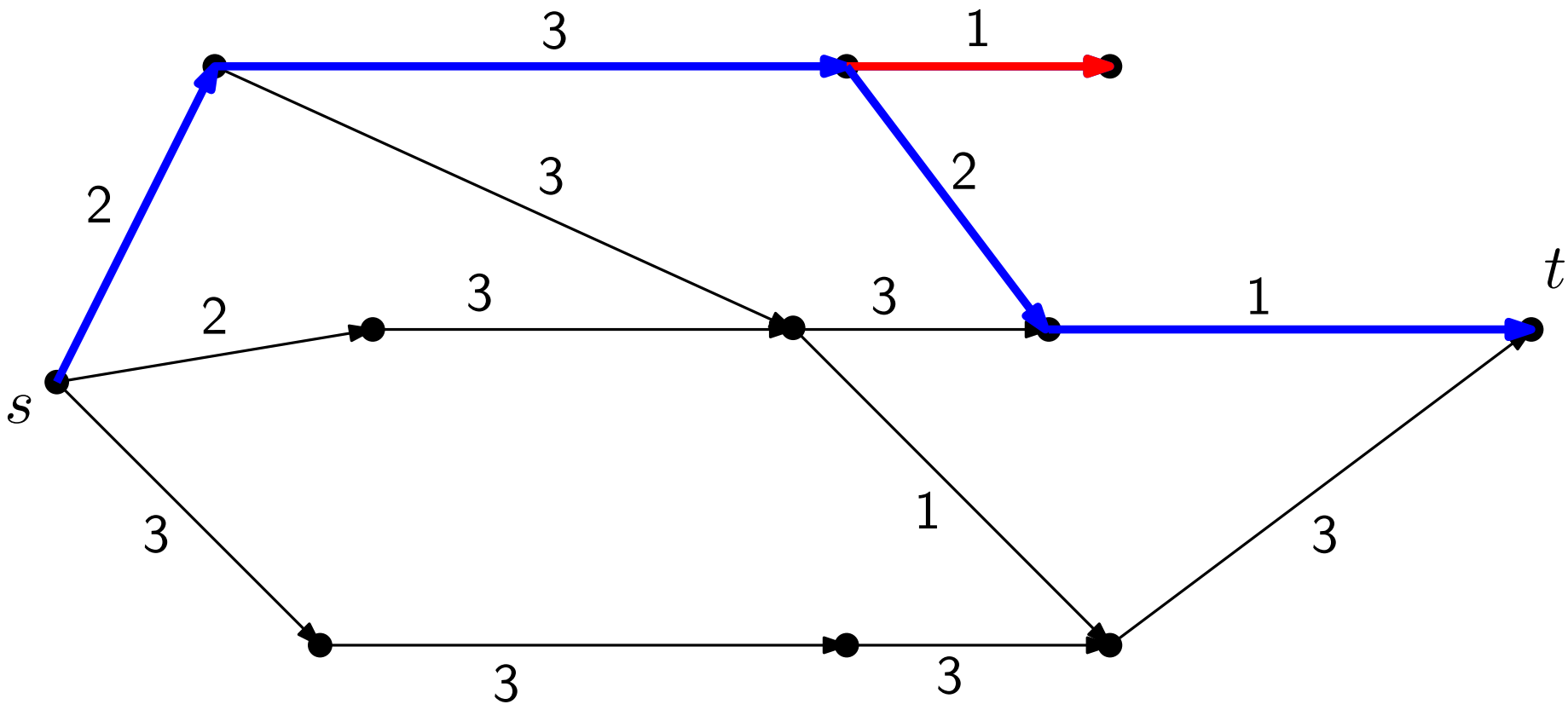
Find an s-t-path with depth-first search.



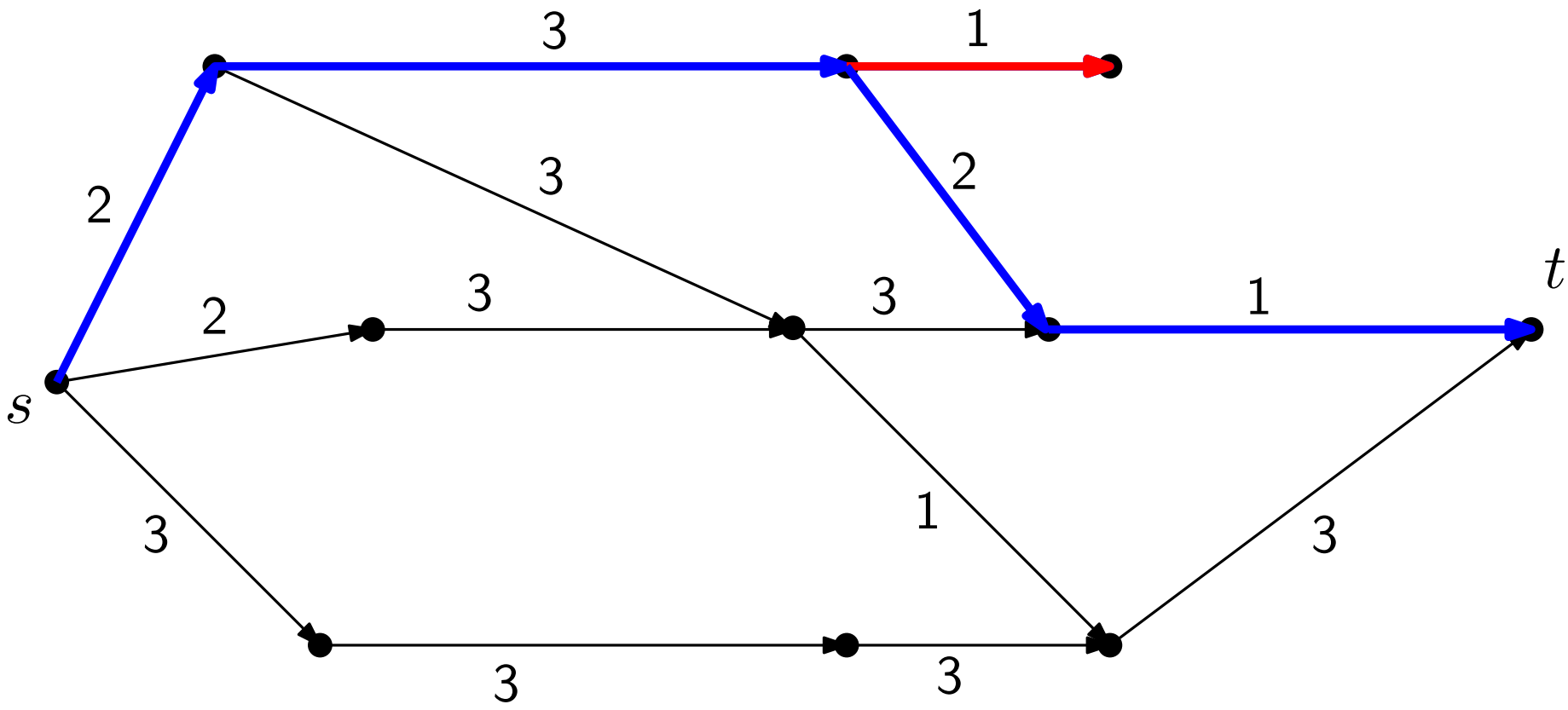
Find an s-t-path with depth-first search.



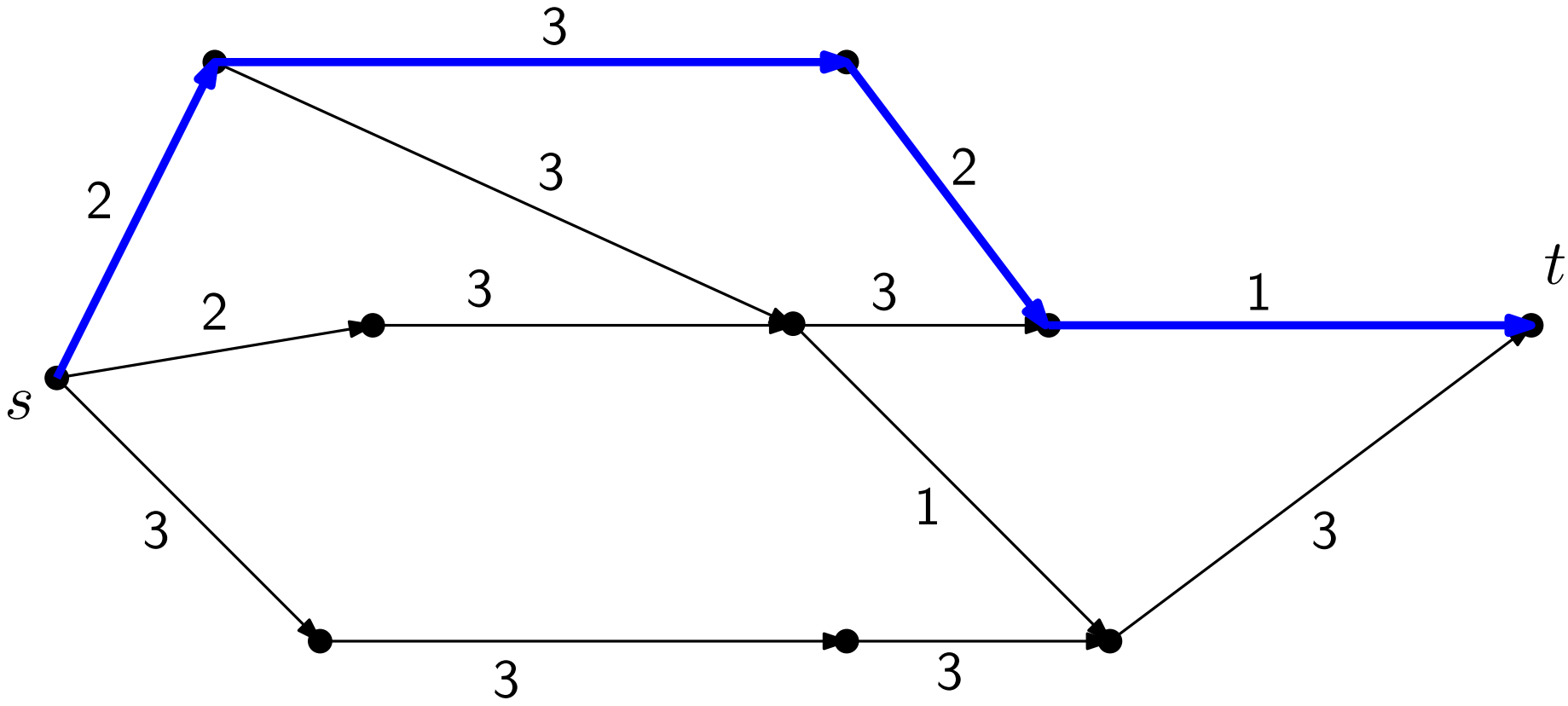
Find an s-t-path with depth-first search.



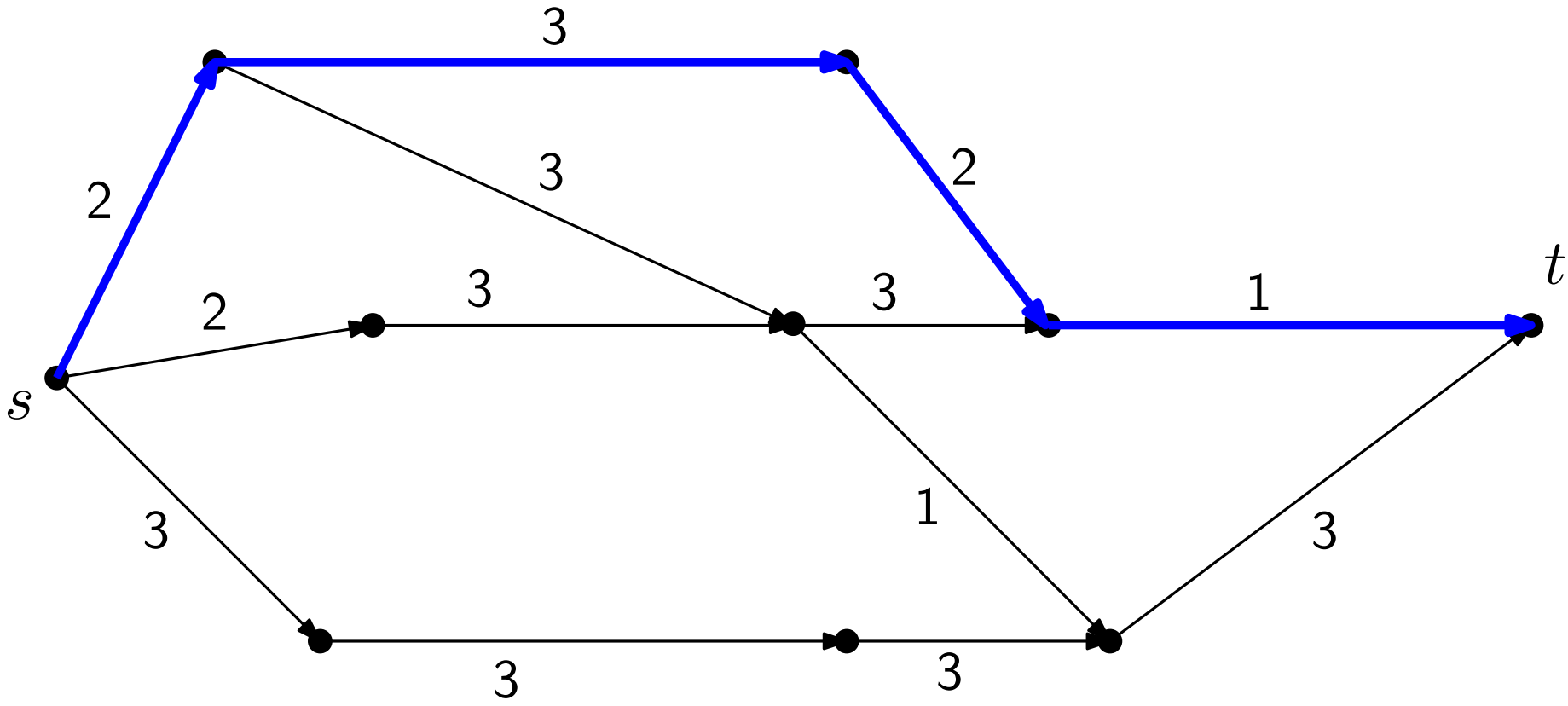
Find an s-t-path with depth-first search.



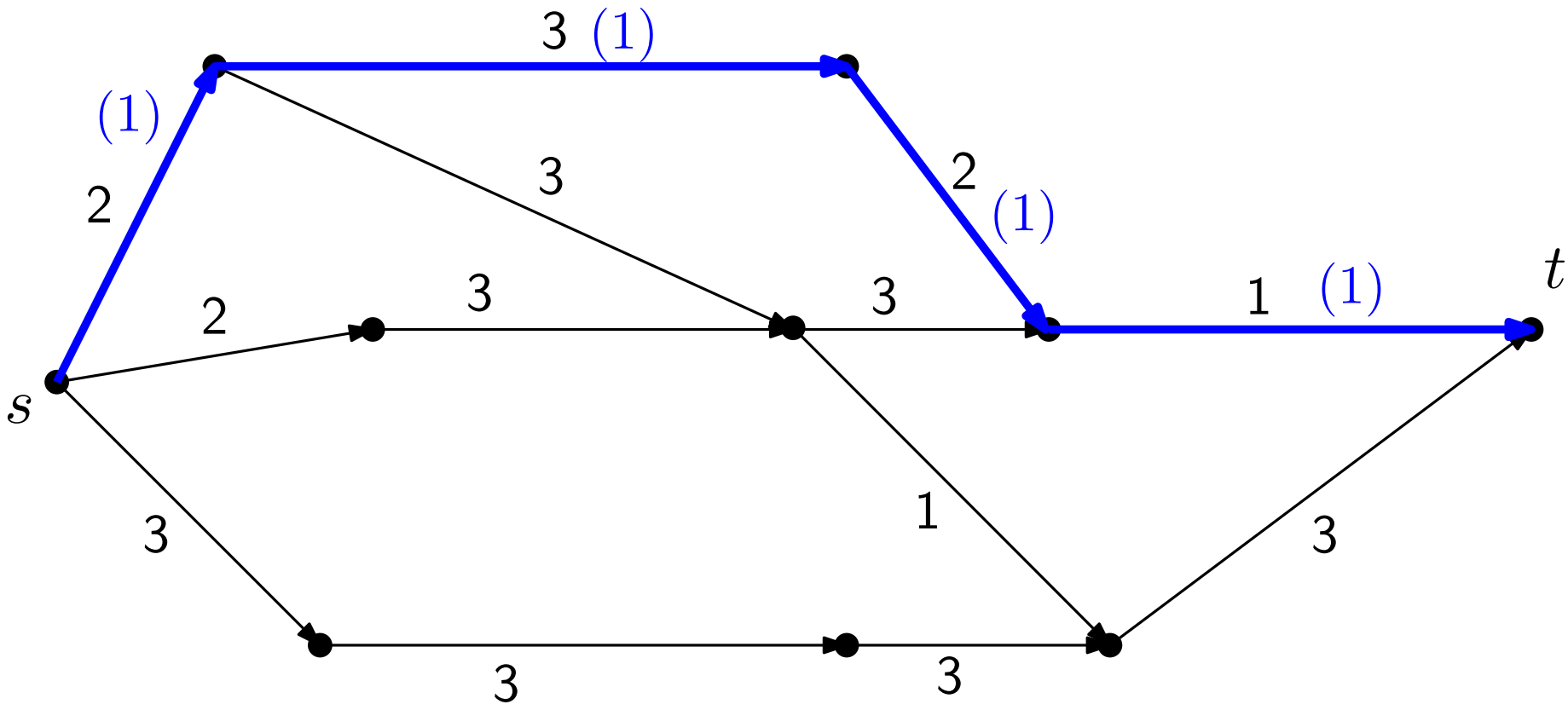
Find an s-t-path with depth-first search.
Delete dead ends.



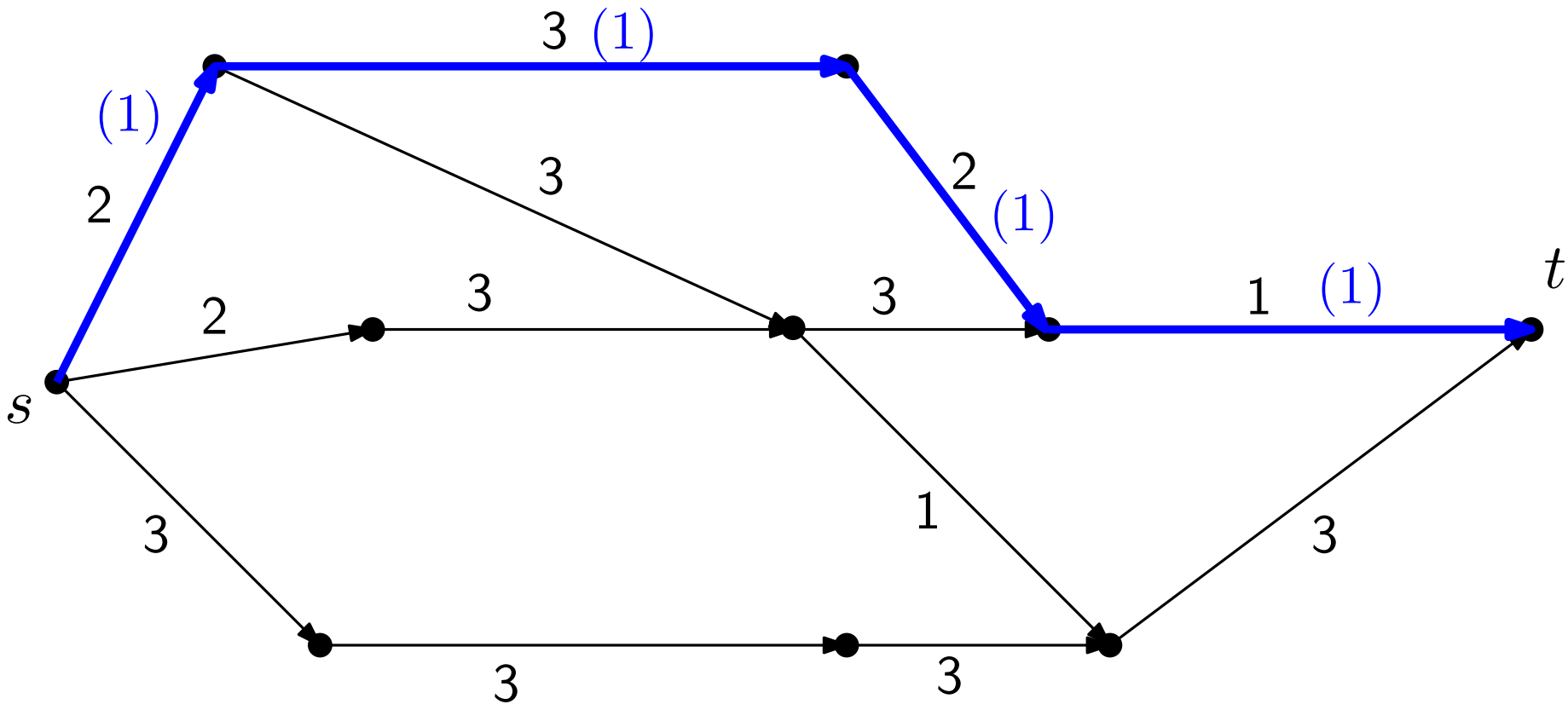
Find an s-t-path with depth-first search.
Delete dead ends.



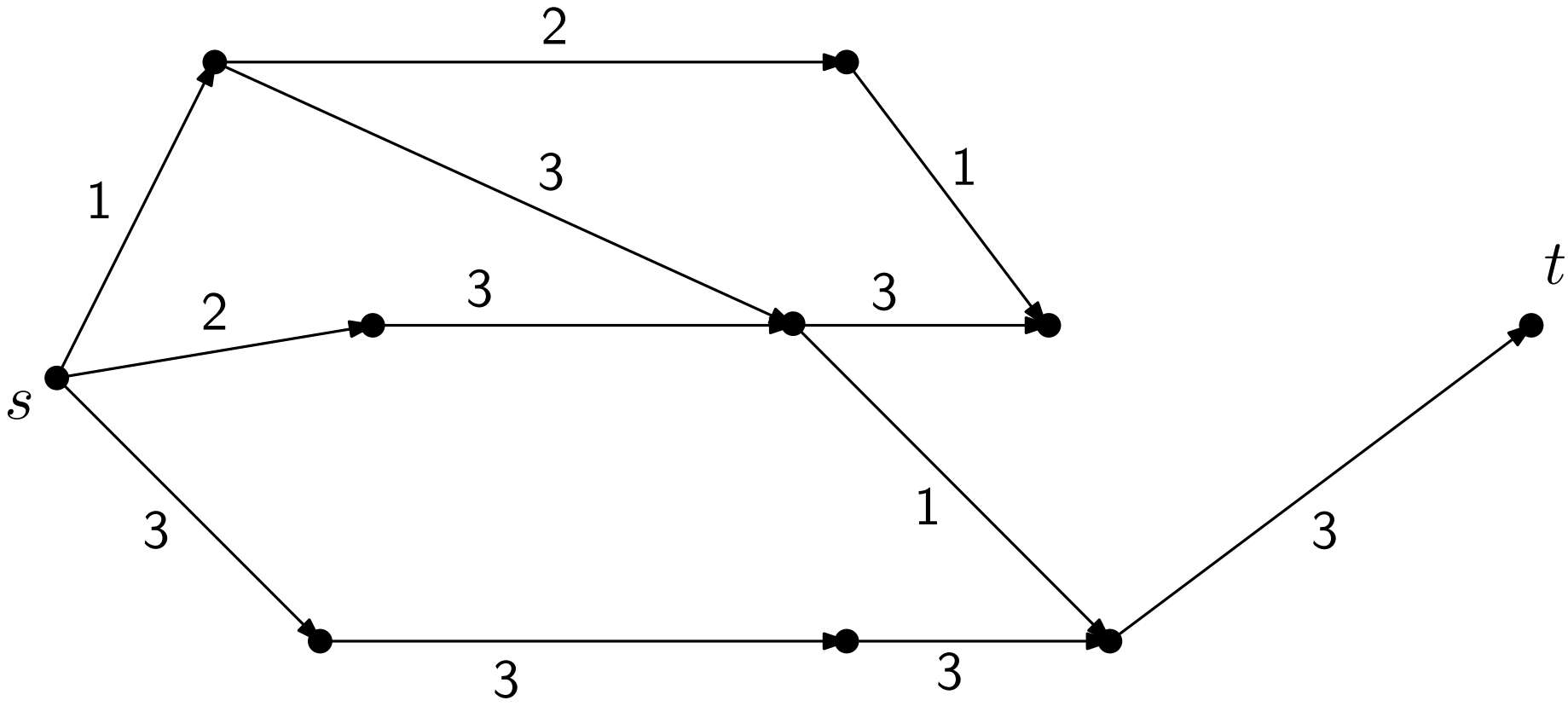
Find an s-t-path with depth-first search.
Delete dead ends.
Route flow.



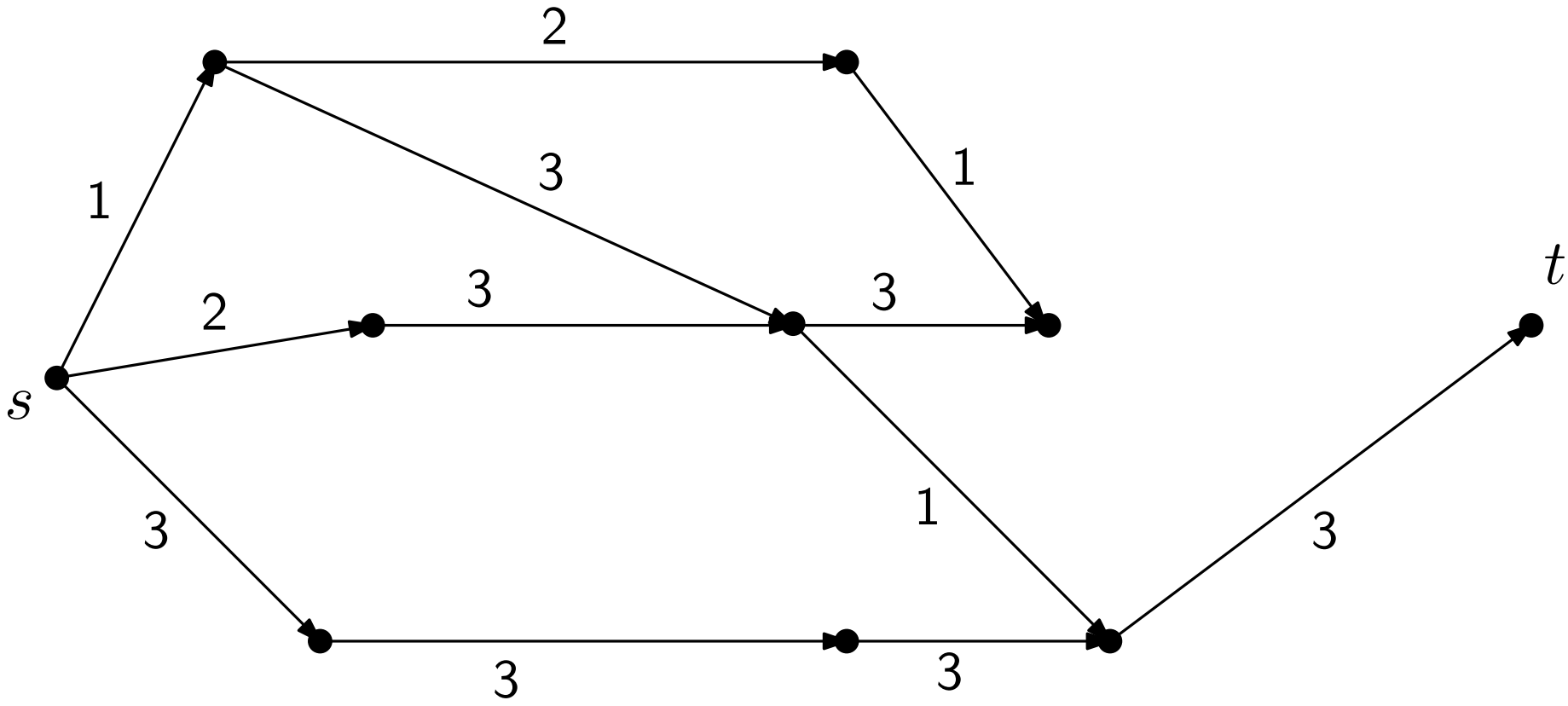
Find an s-t-path with depth-first search.
Delete dead ends.
Route flow.



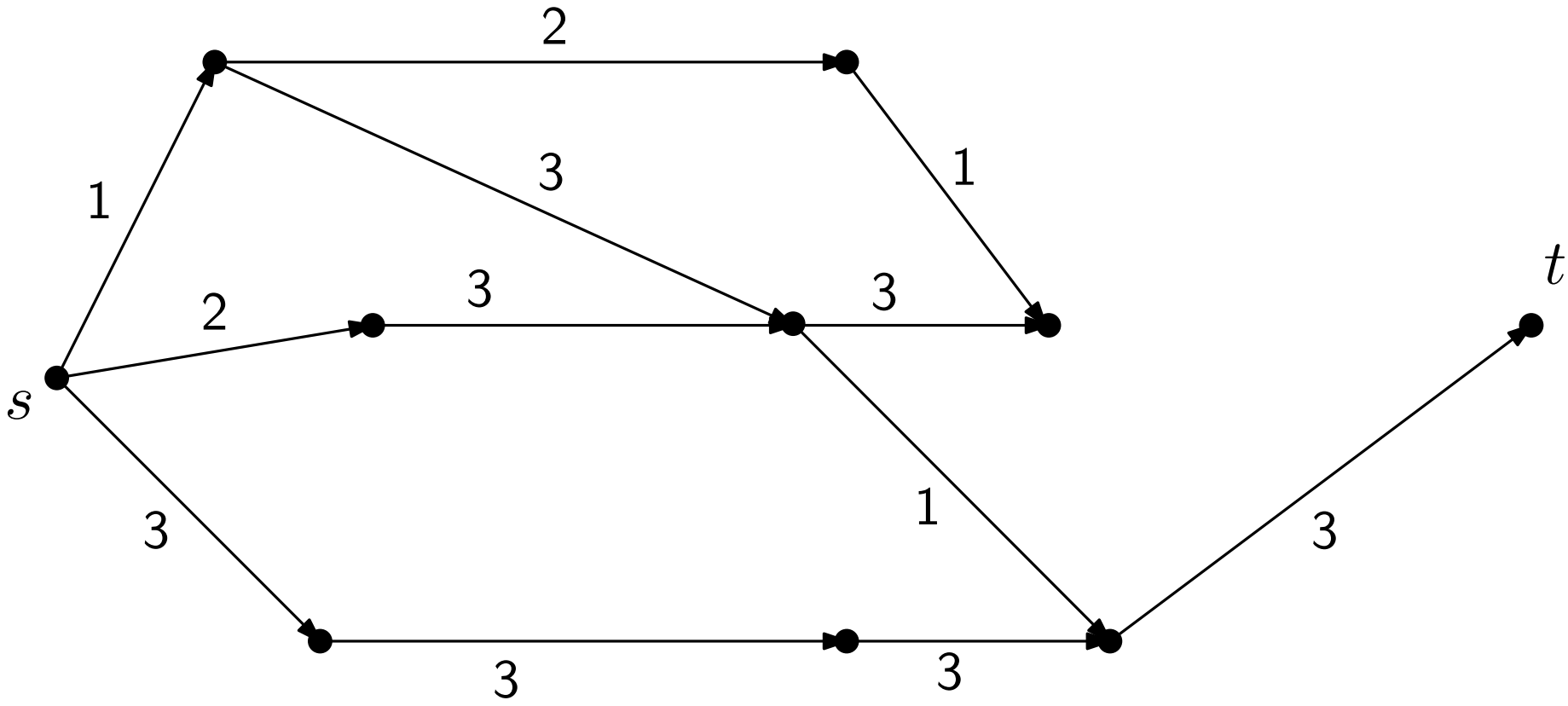
Find an s-t-path with depth-first search.
Delete dead ends.
Route flow.
Update capacities.



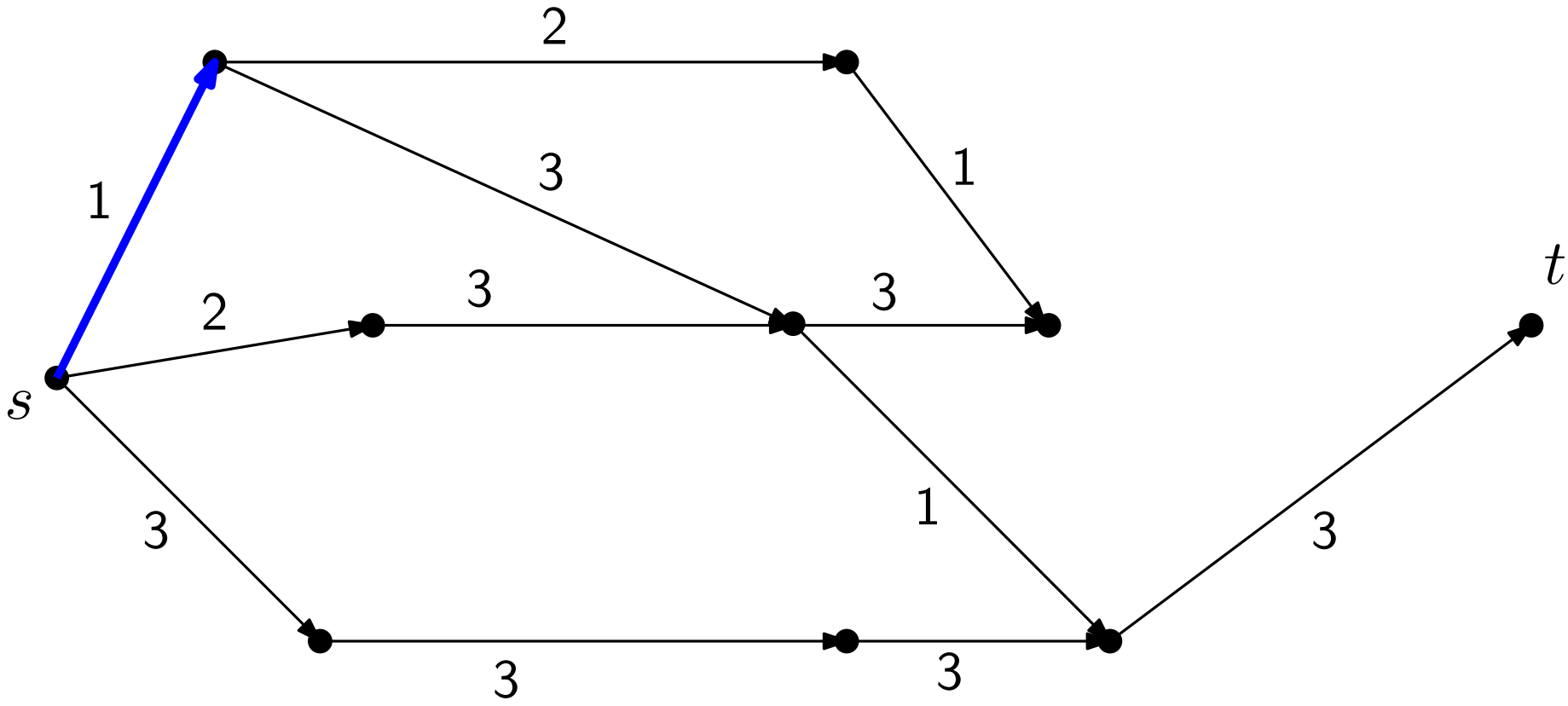
Find an s-t-path with depth-first search.
Delete dead ends.
Route flow.
Update capacities.



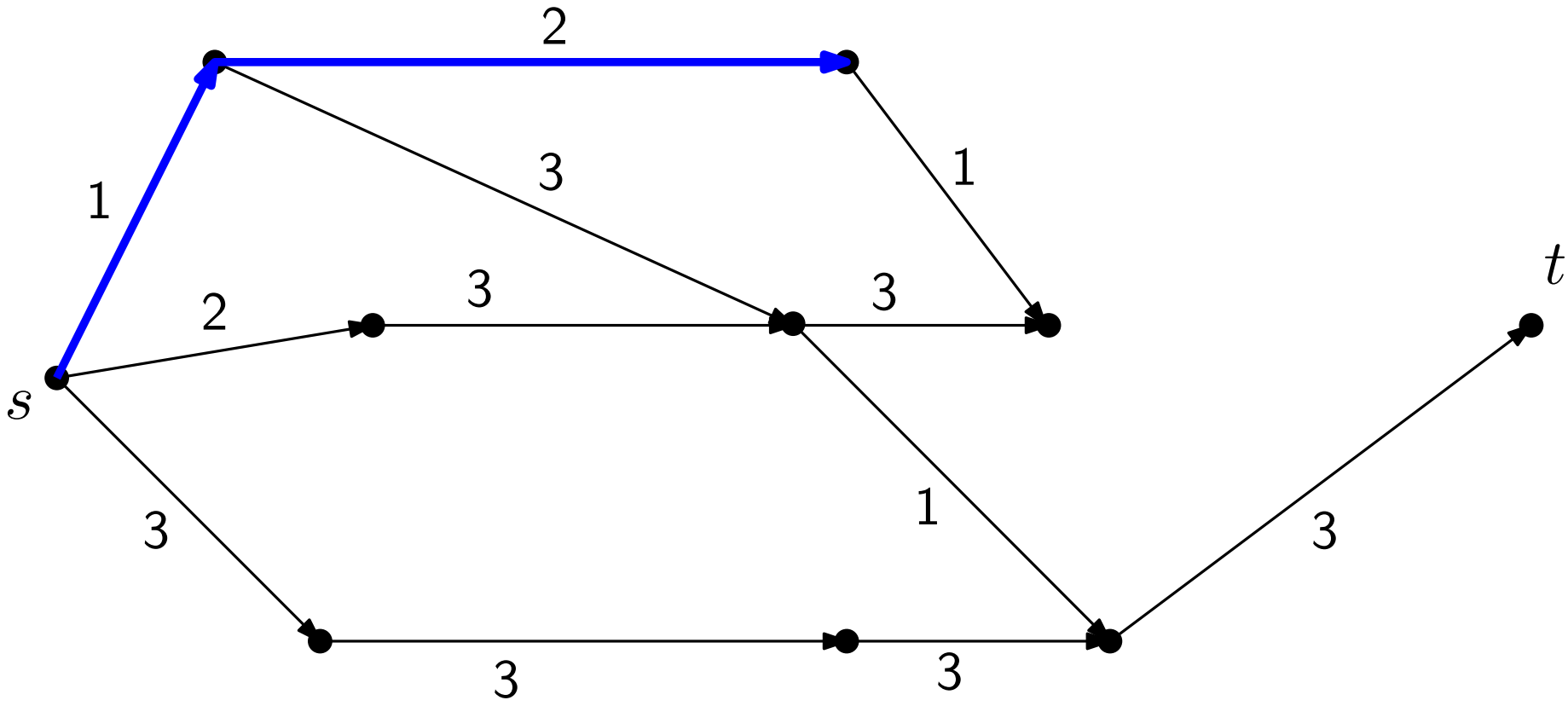
Find an s-t-path with depth-first search.
Delete dead ends.
Route flow.
Update capacities. Repeat!



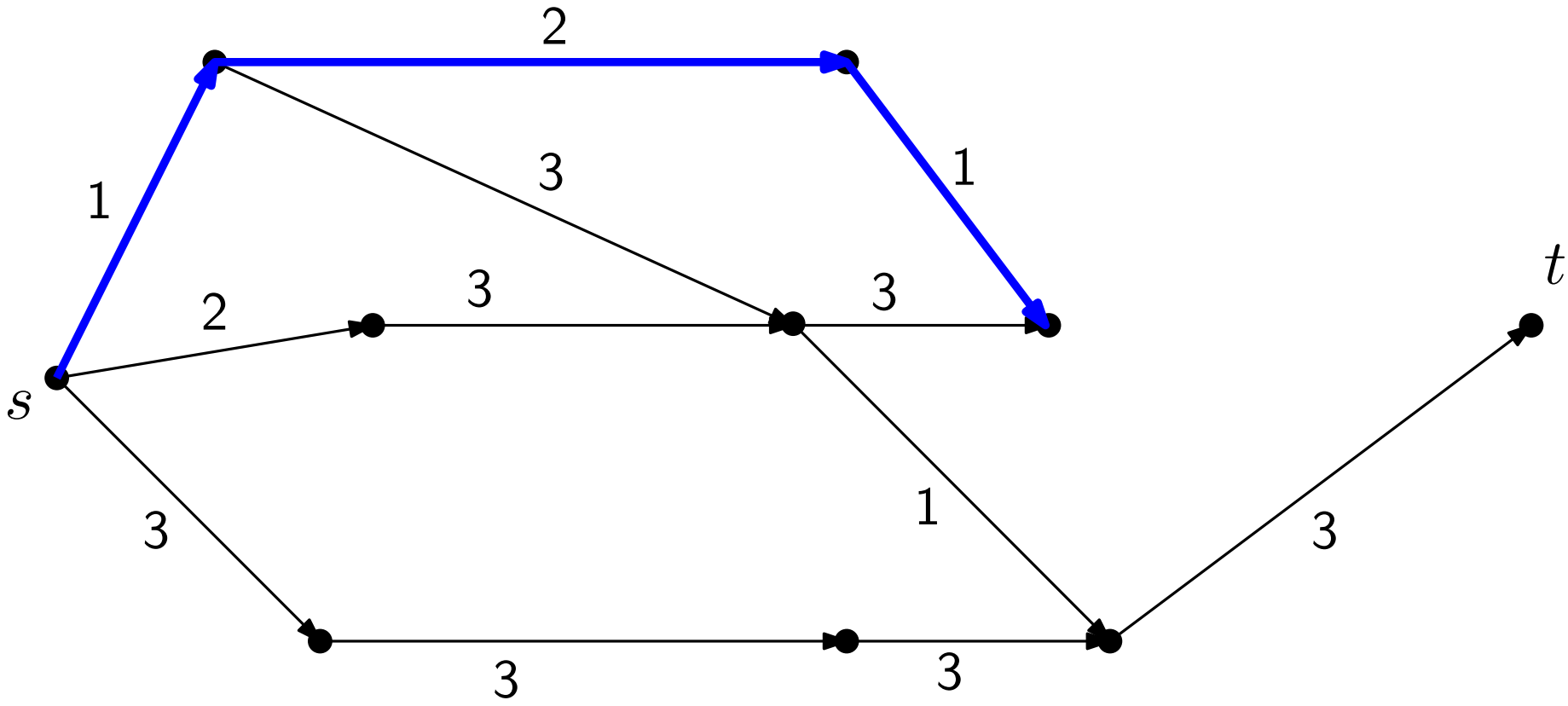
Find an s-t-path with depth-first search.



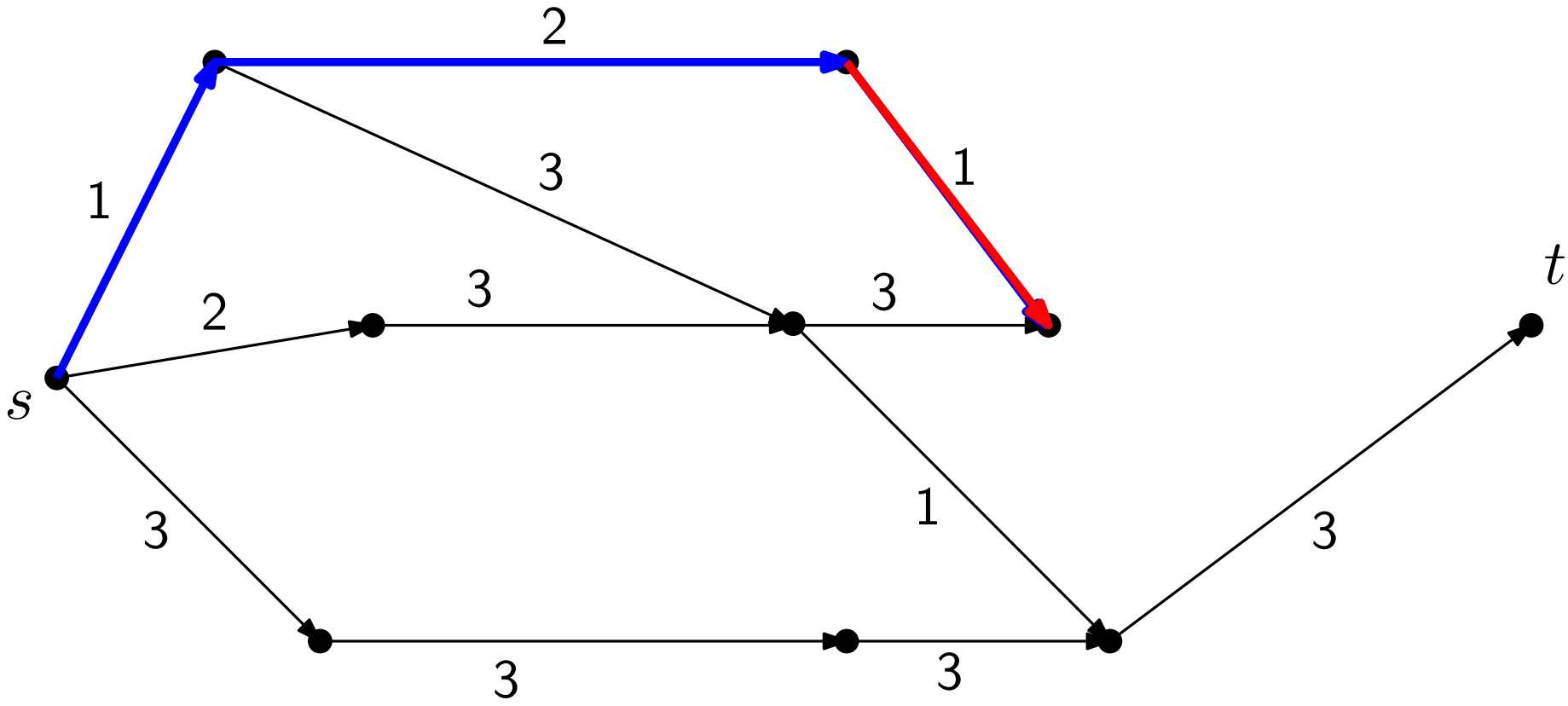
Find an s-t-path with depth-first search.



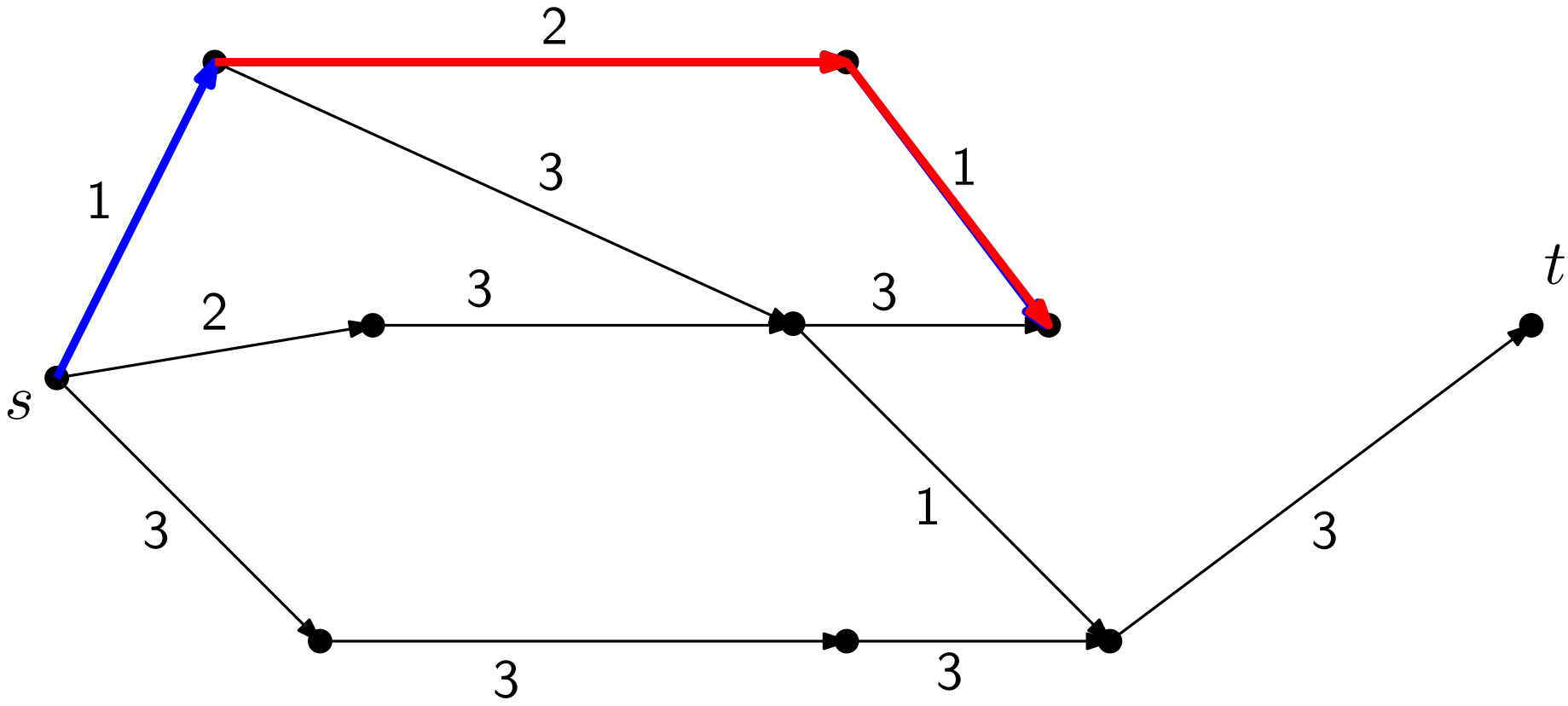
Find an s-t-path with depth-first search.



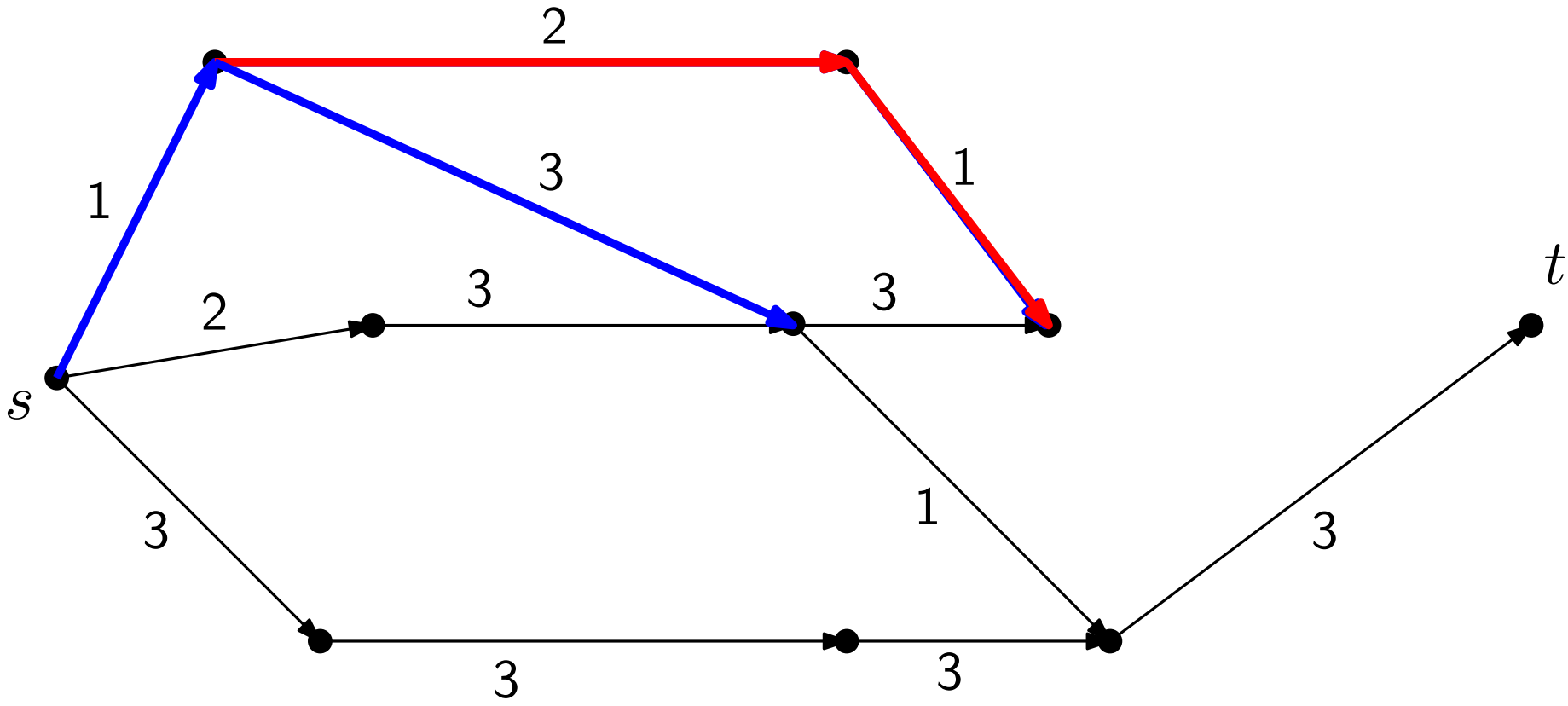
Find an s-t-path with depth-first search.



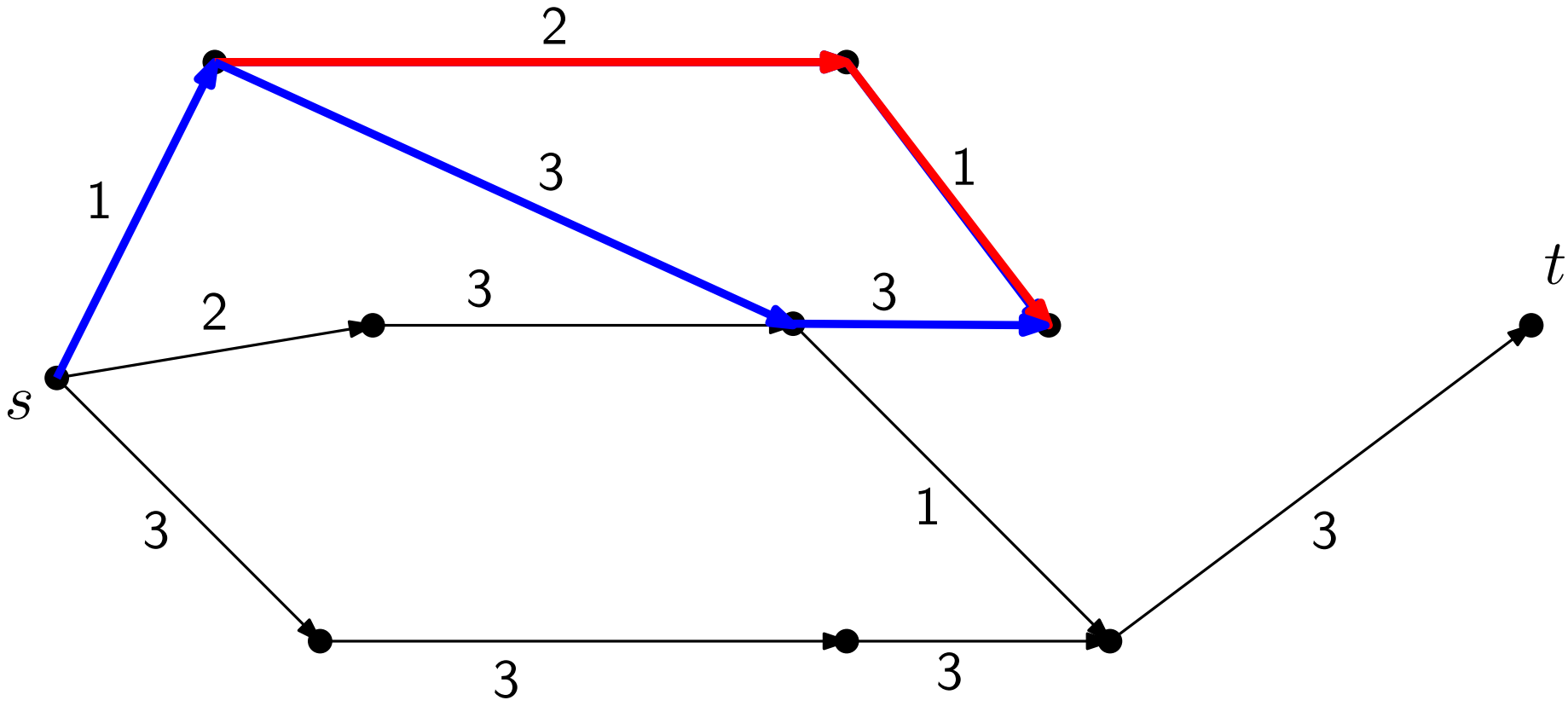
Find an s-t-path with depth-first search.



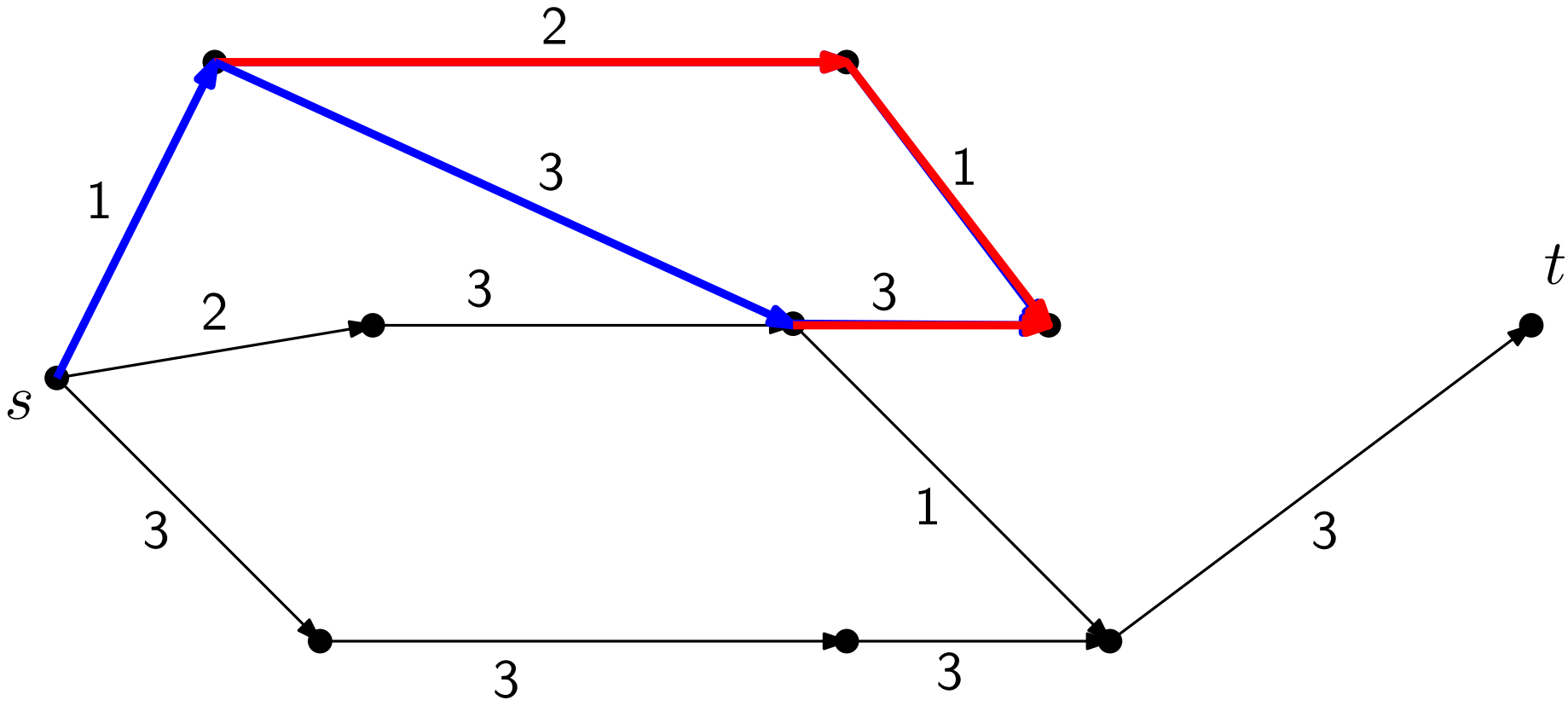
Find an s-t-path with depth-first search.



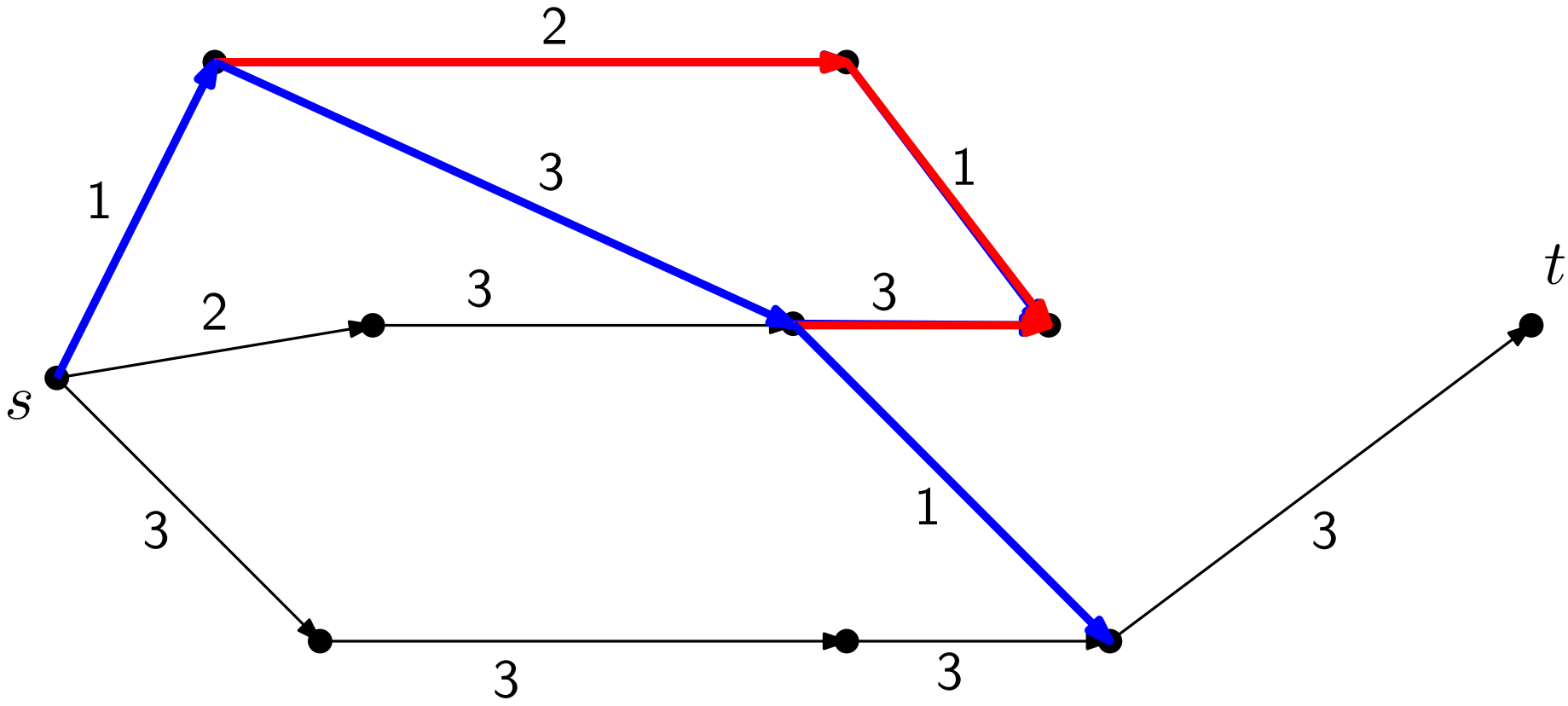
Find an s-t-path with depth-first search.



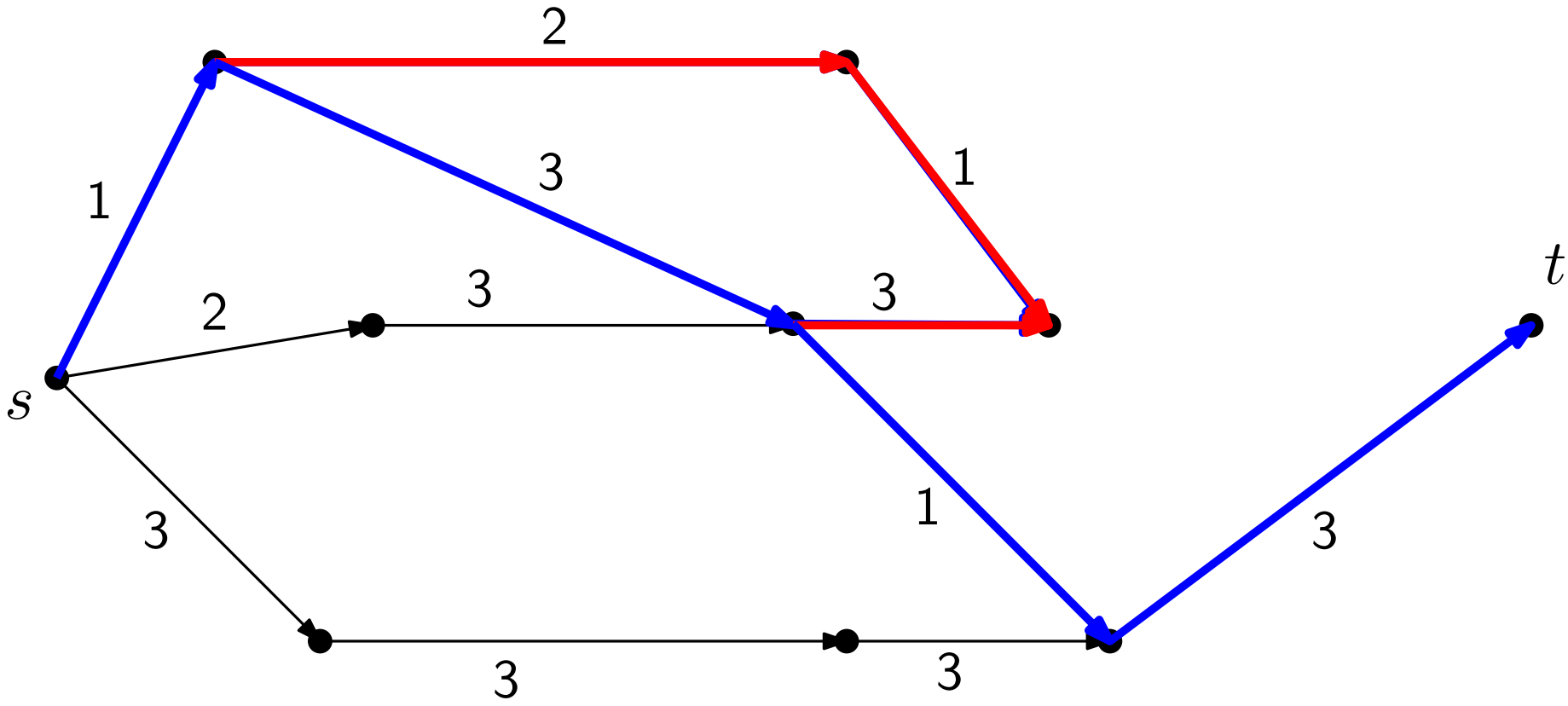
Find an s-t-path with depth-first search.



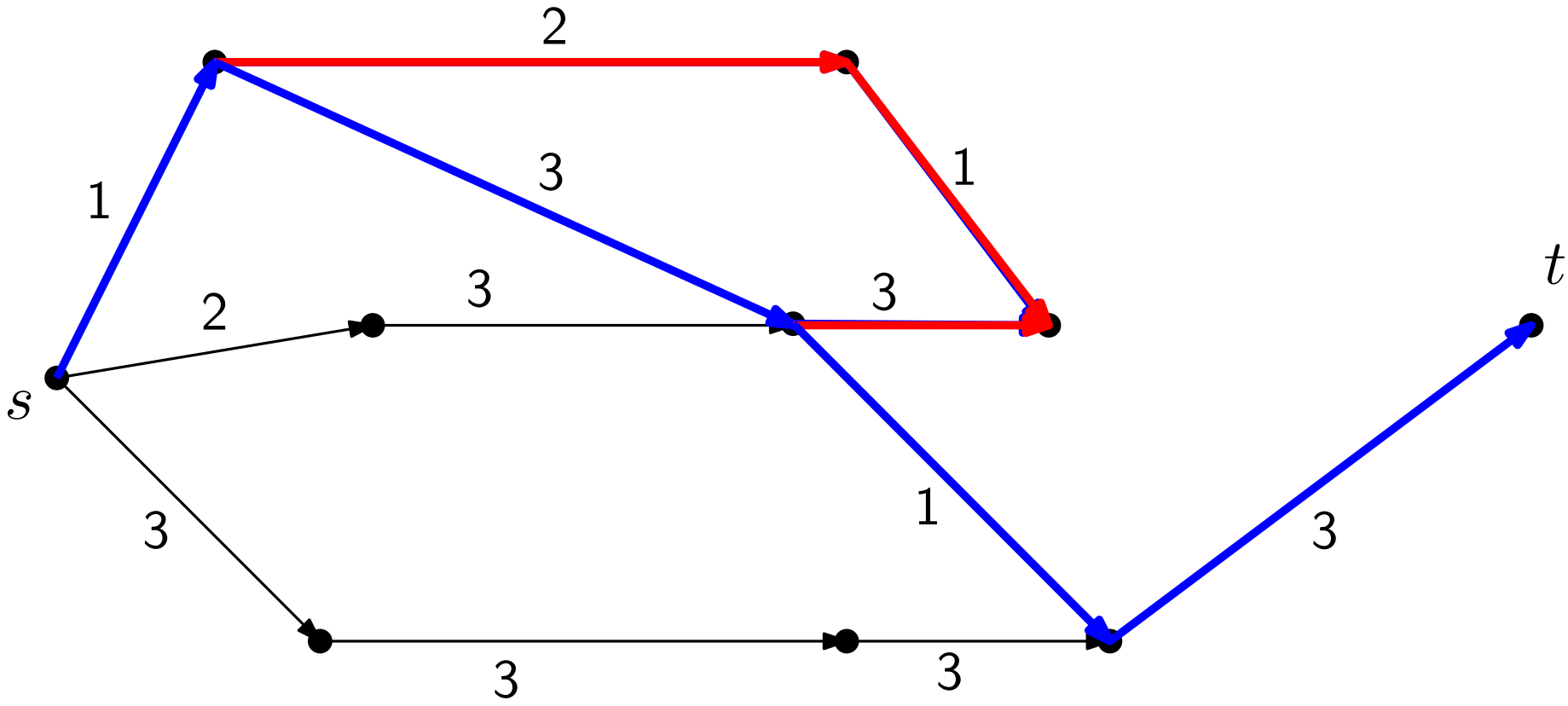
Find an s-t-path with depth-first search.



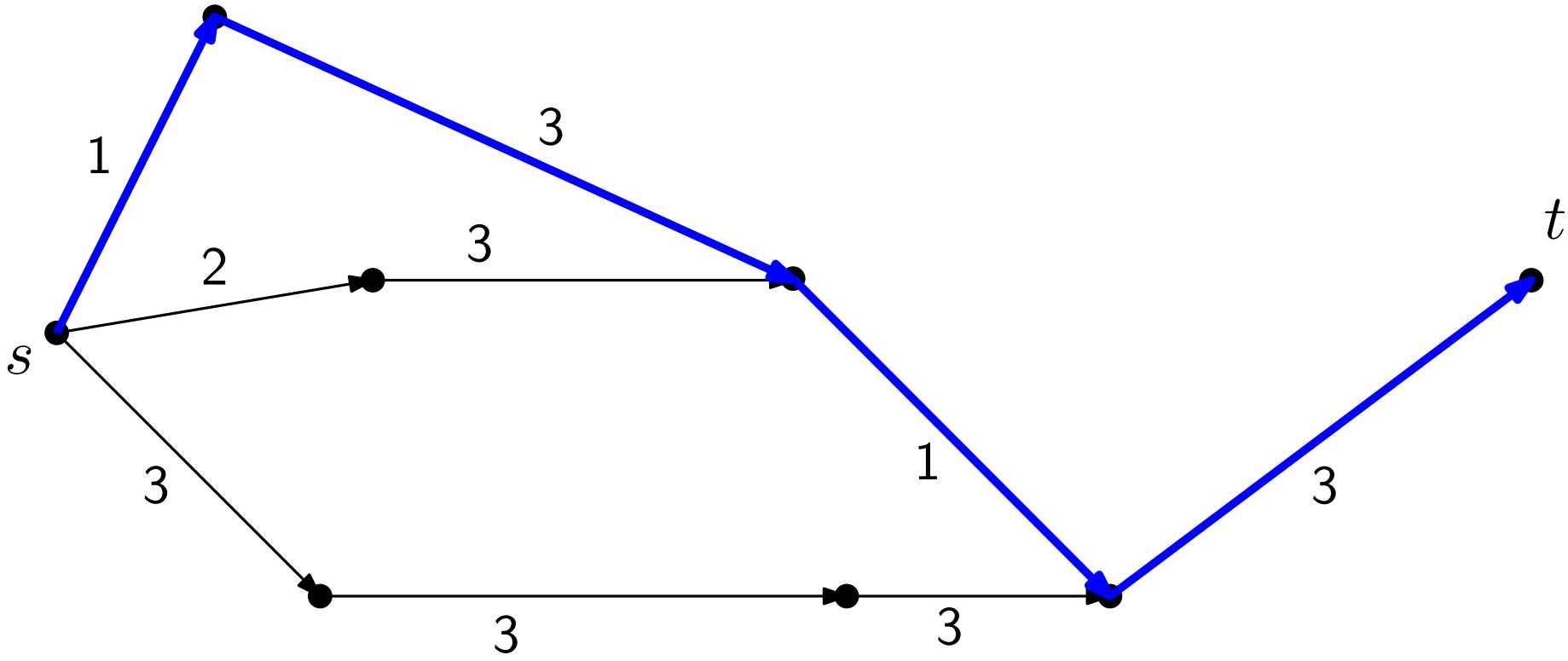
Find an s-t-path with depth-first search.



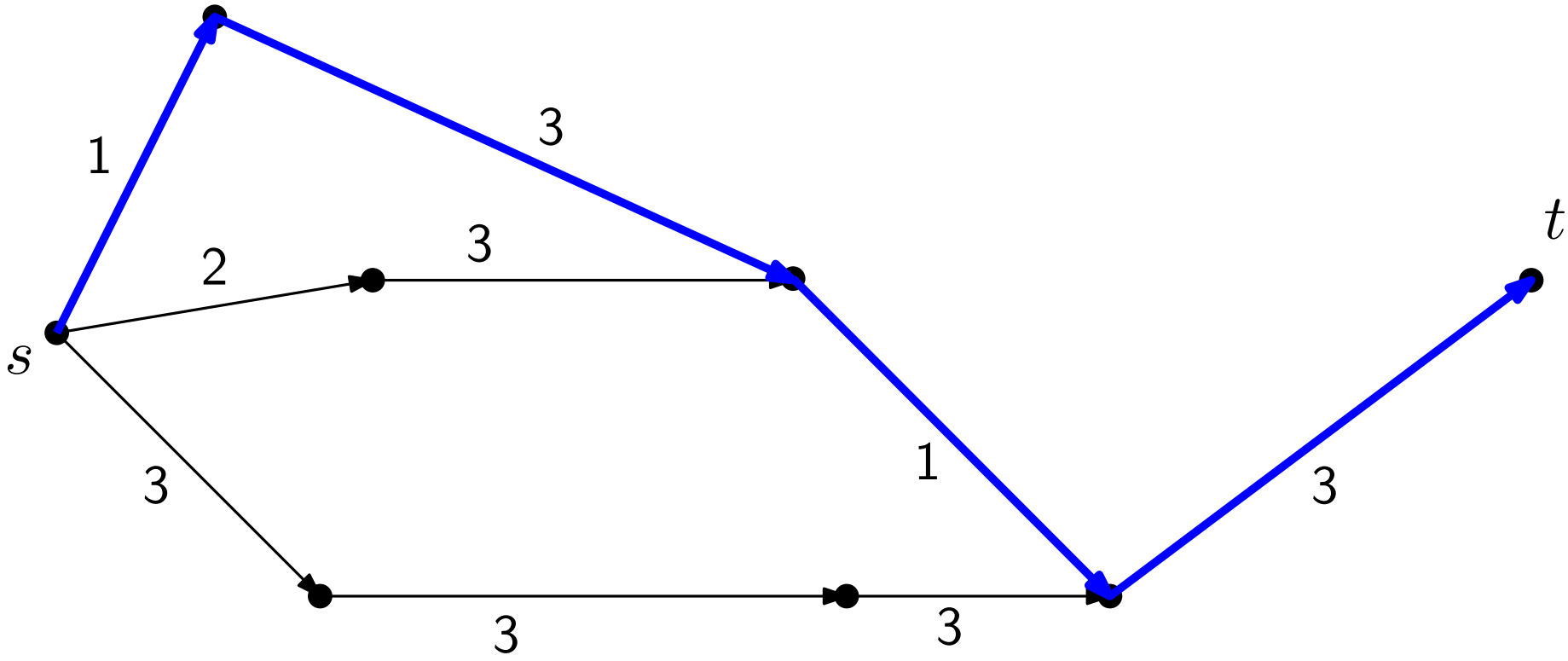
Find an s-t-path with depth-first search.



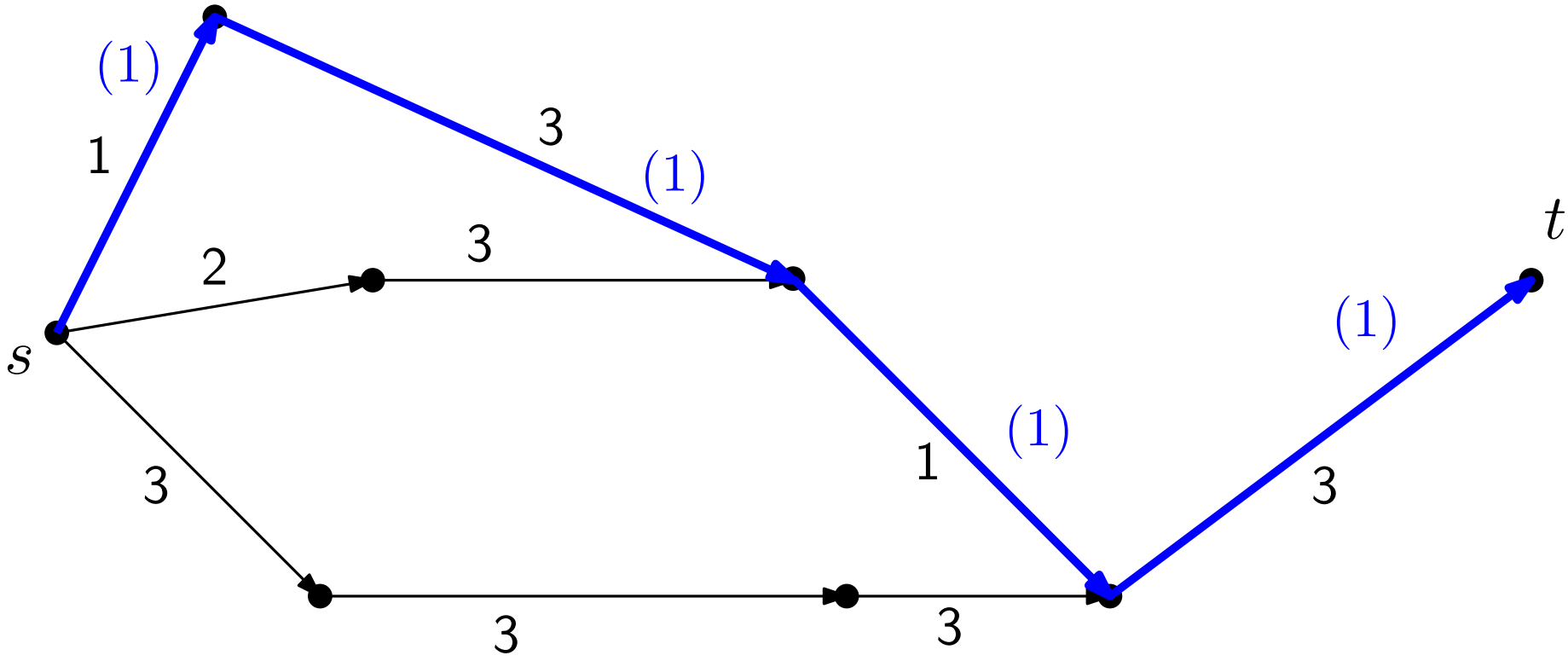
Find an s-t-path with depth-first search.
Delete dead ends.



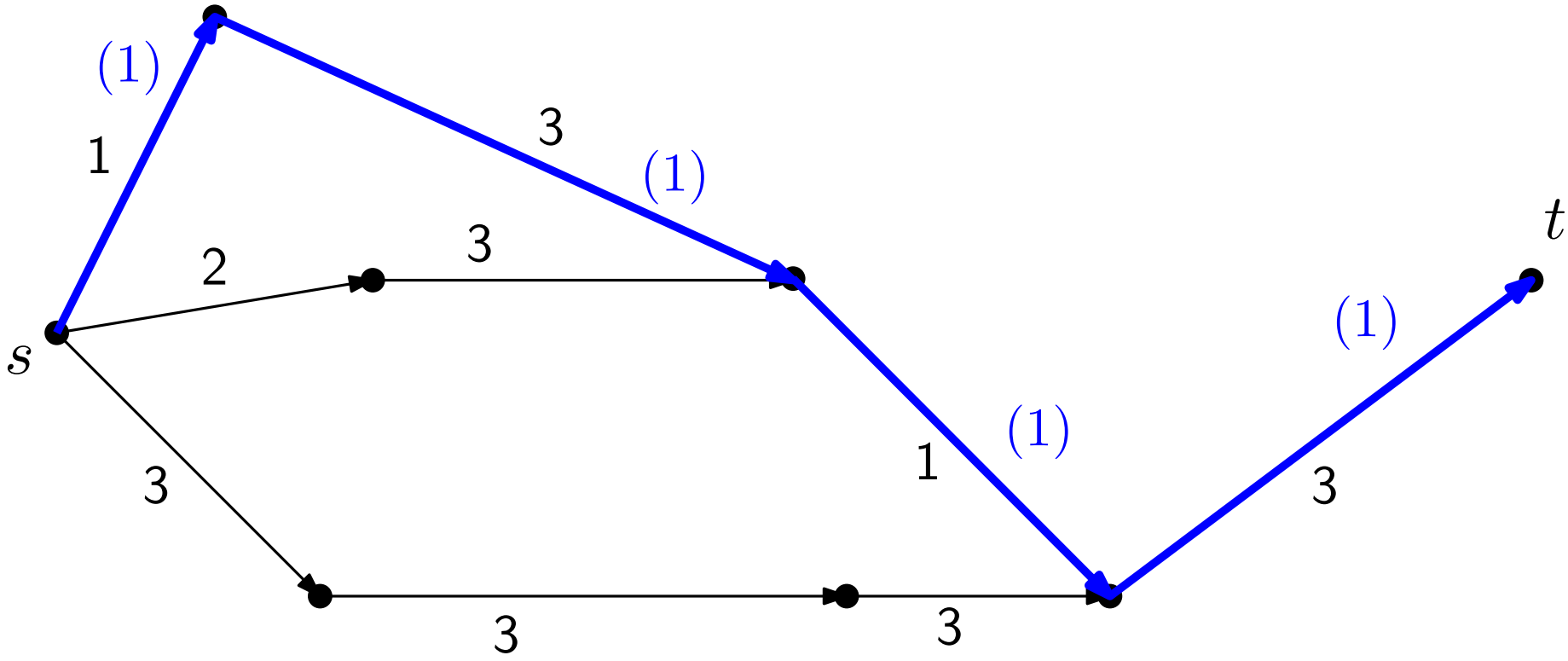
Find an s-t-path with depth-first search.
Delete dead ends.



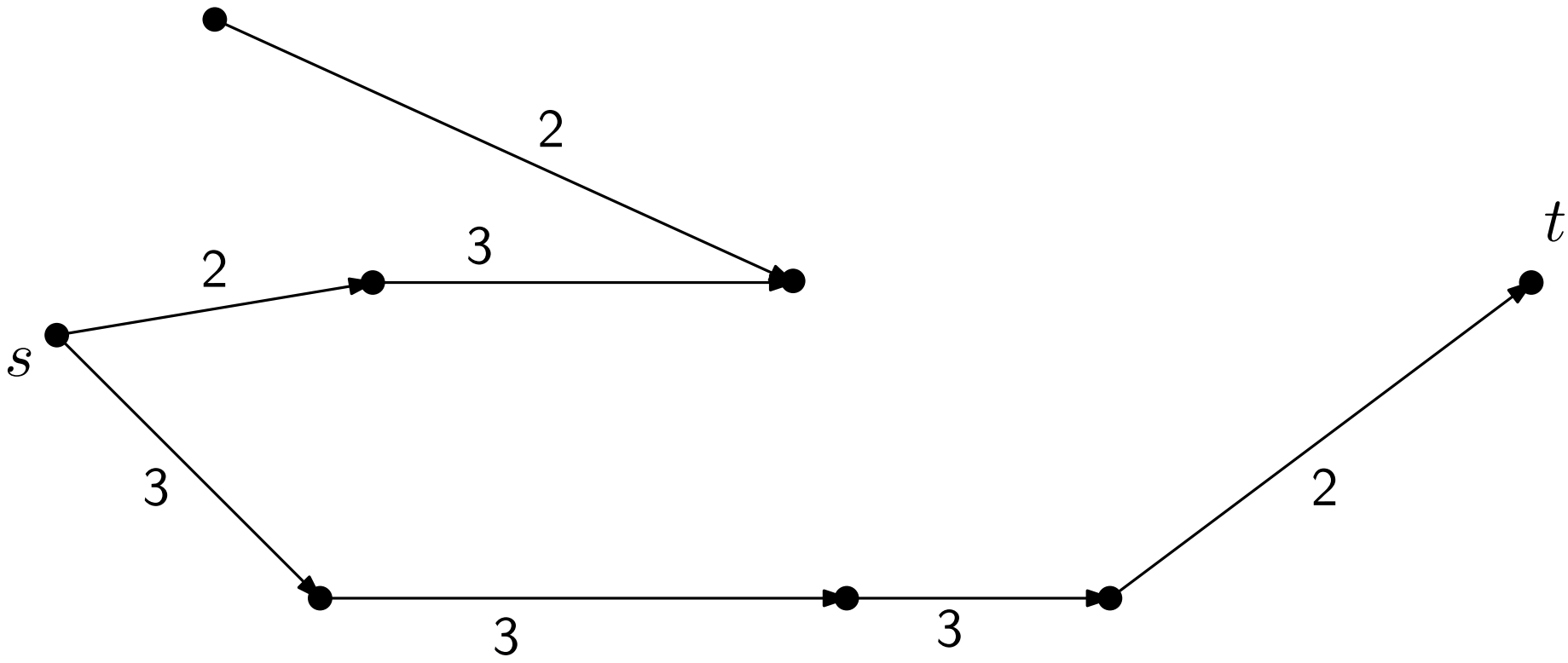
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.



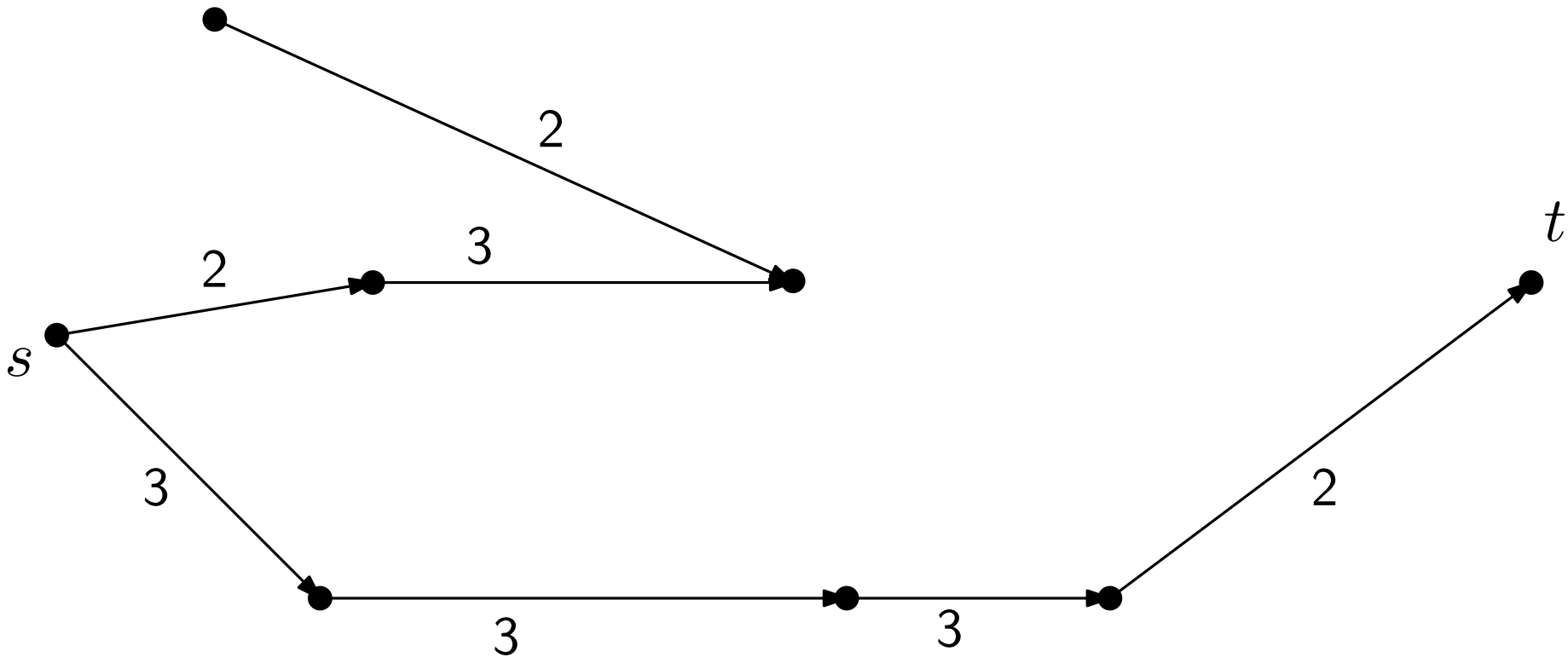
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.



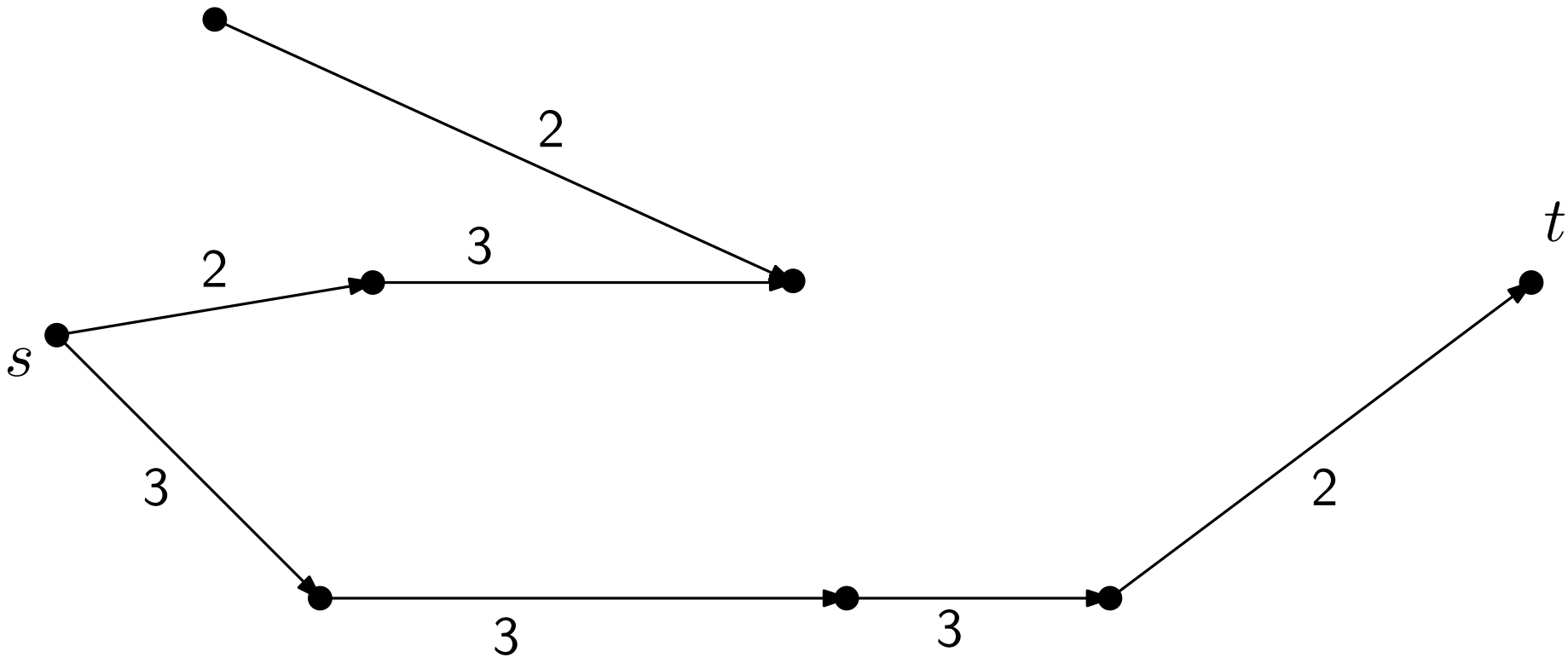
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.
Update capacities.



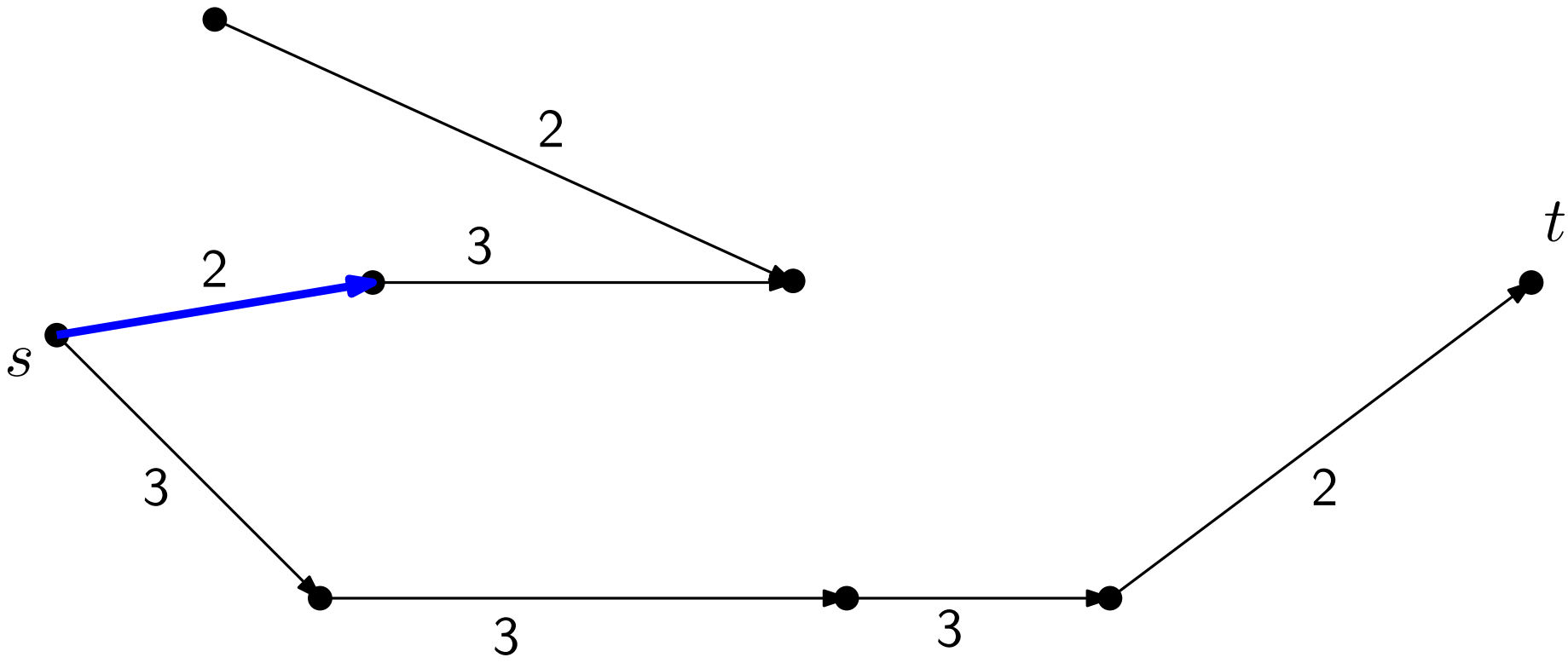
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.
Update capacities.



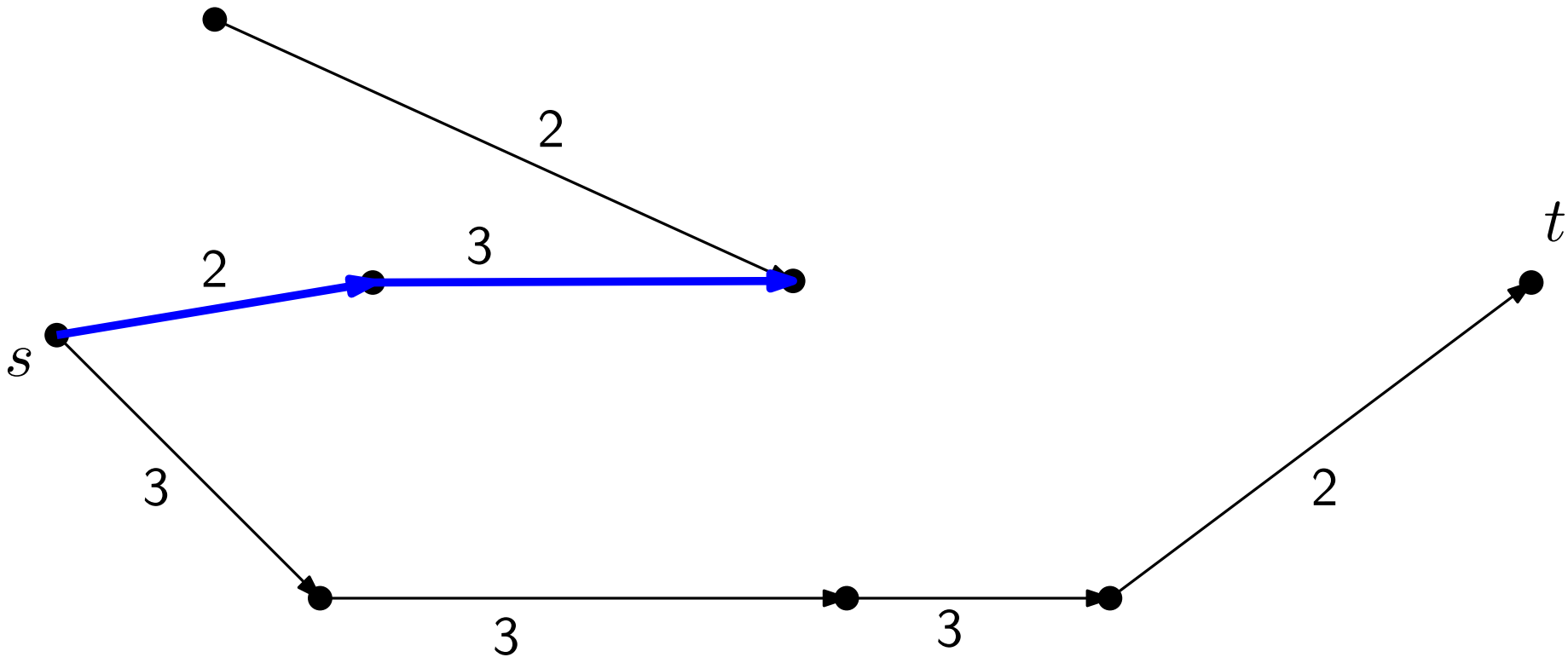
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.
Update capacities. Repeat!



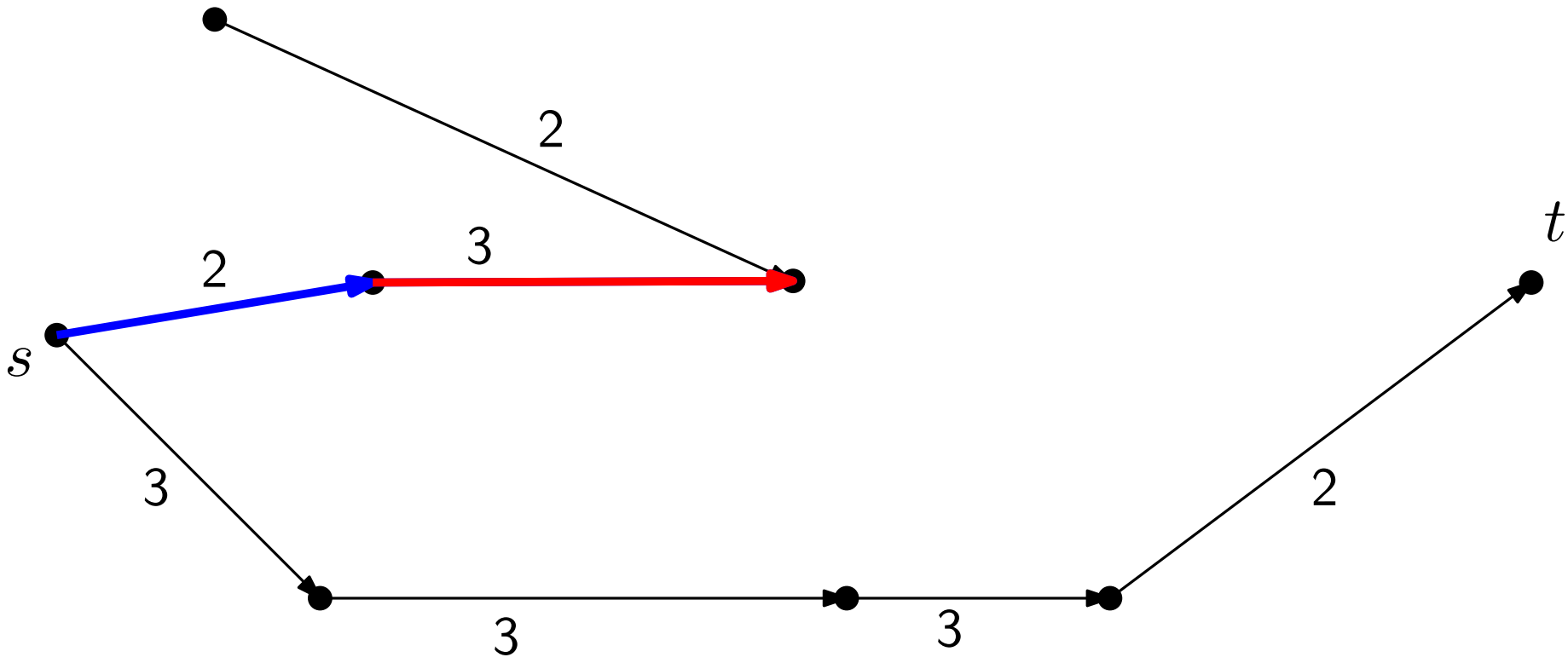
Find an s-t-path with depth-first search.



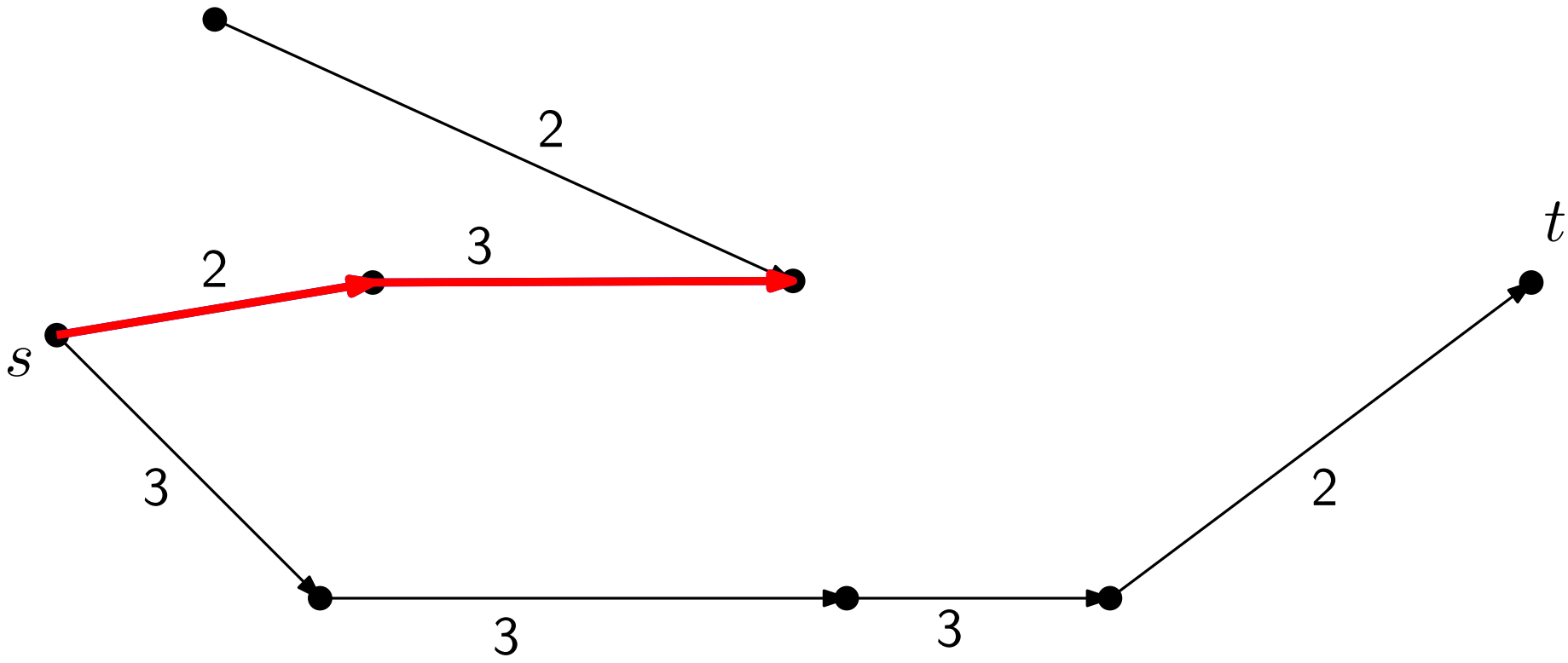
Find an s-t-path with depth-first search.



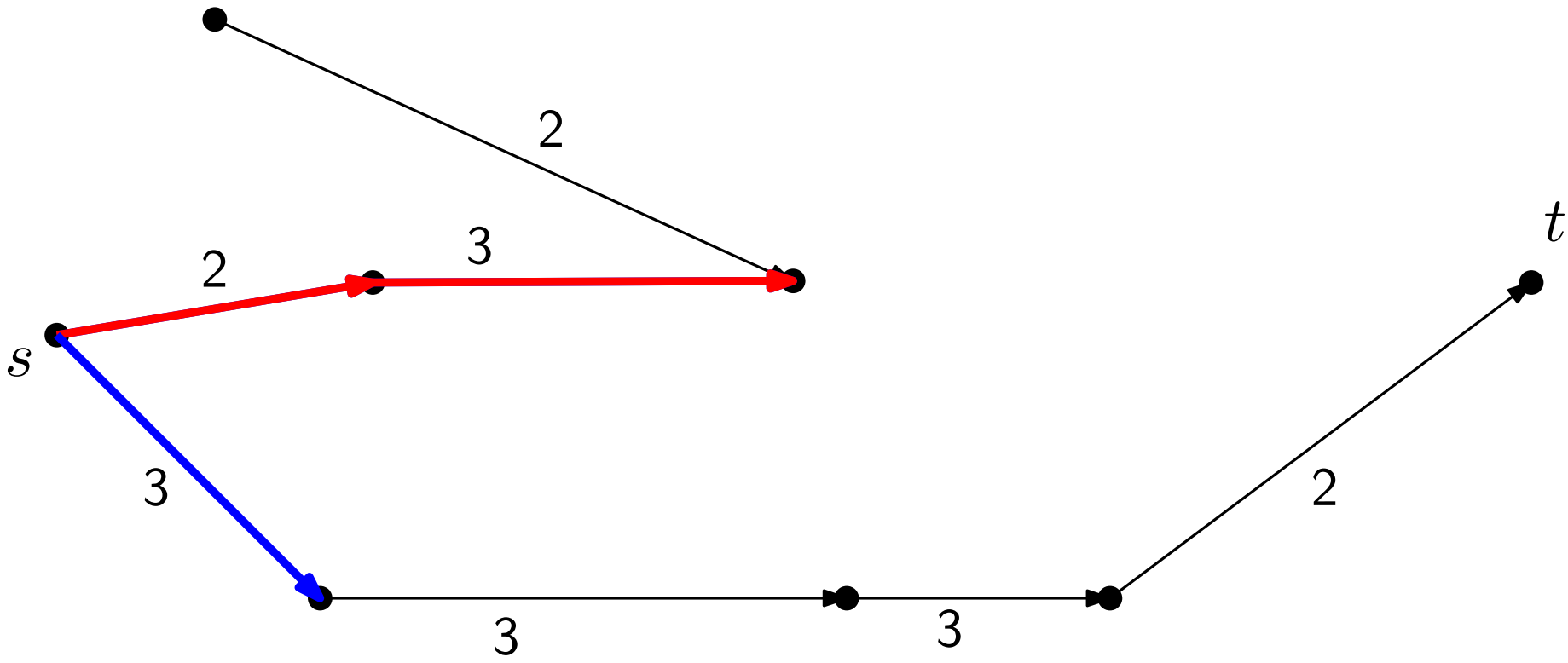
Find an s-t-path with depth-first search.



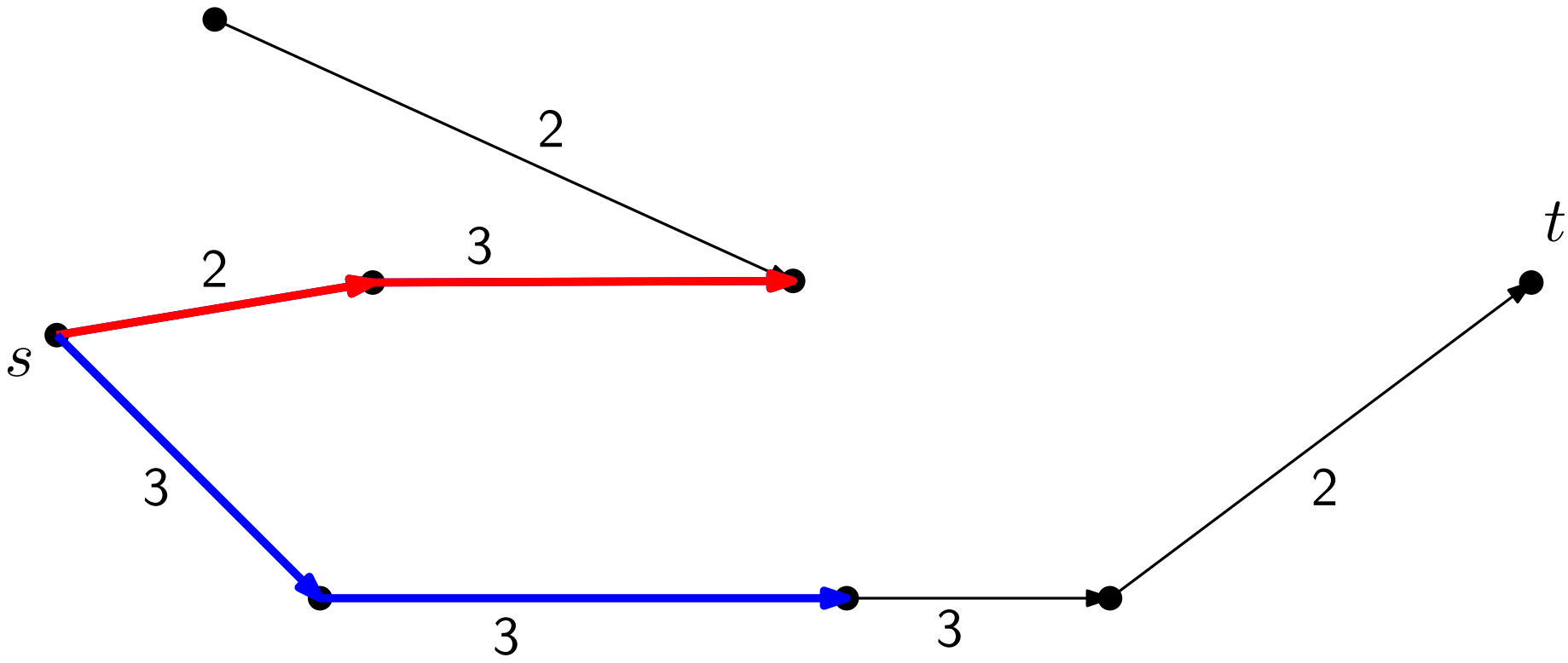
Find an s-t-path with depth-first search.



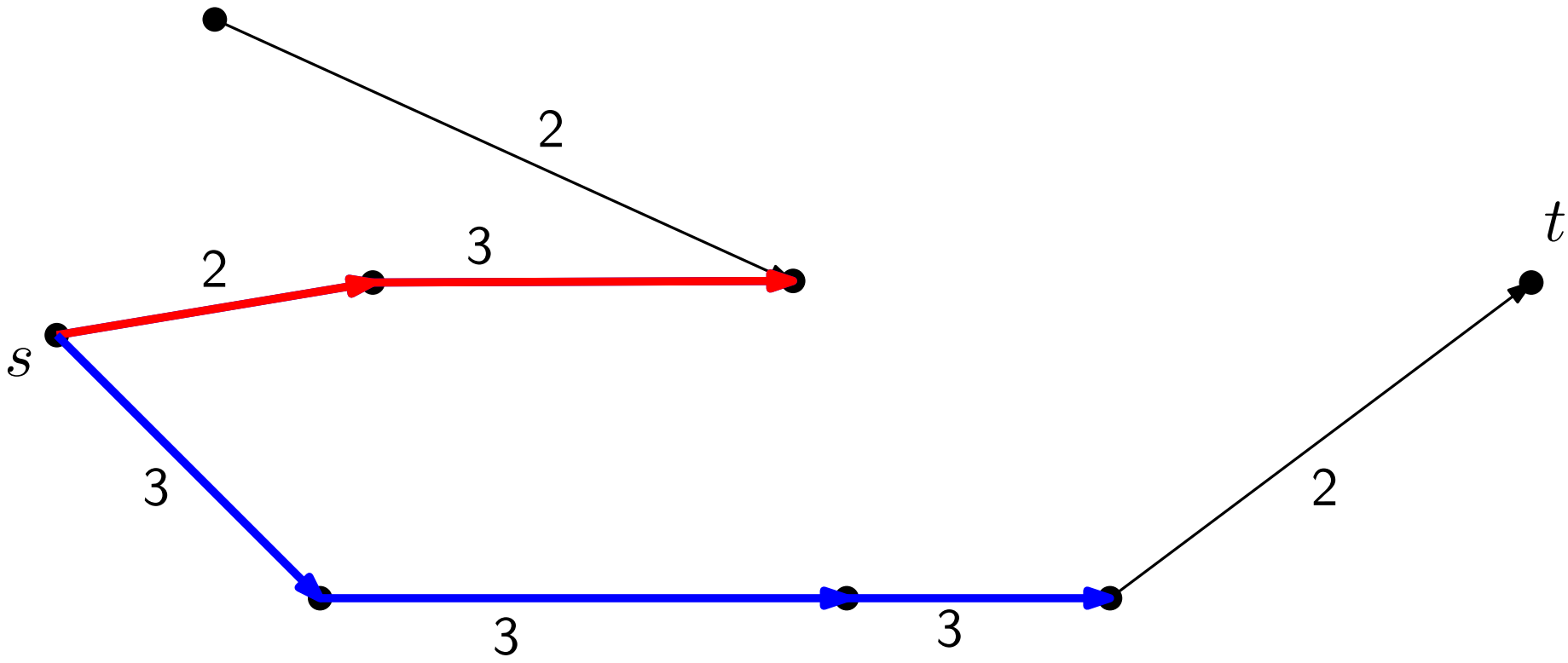
Find an s-t-path with depth-first search.



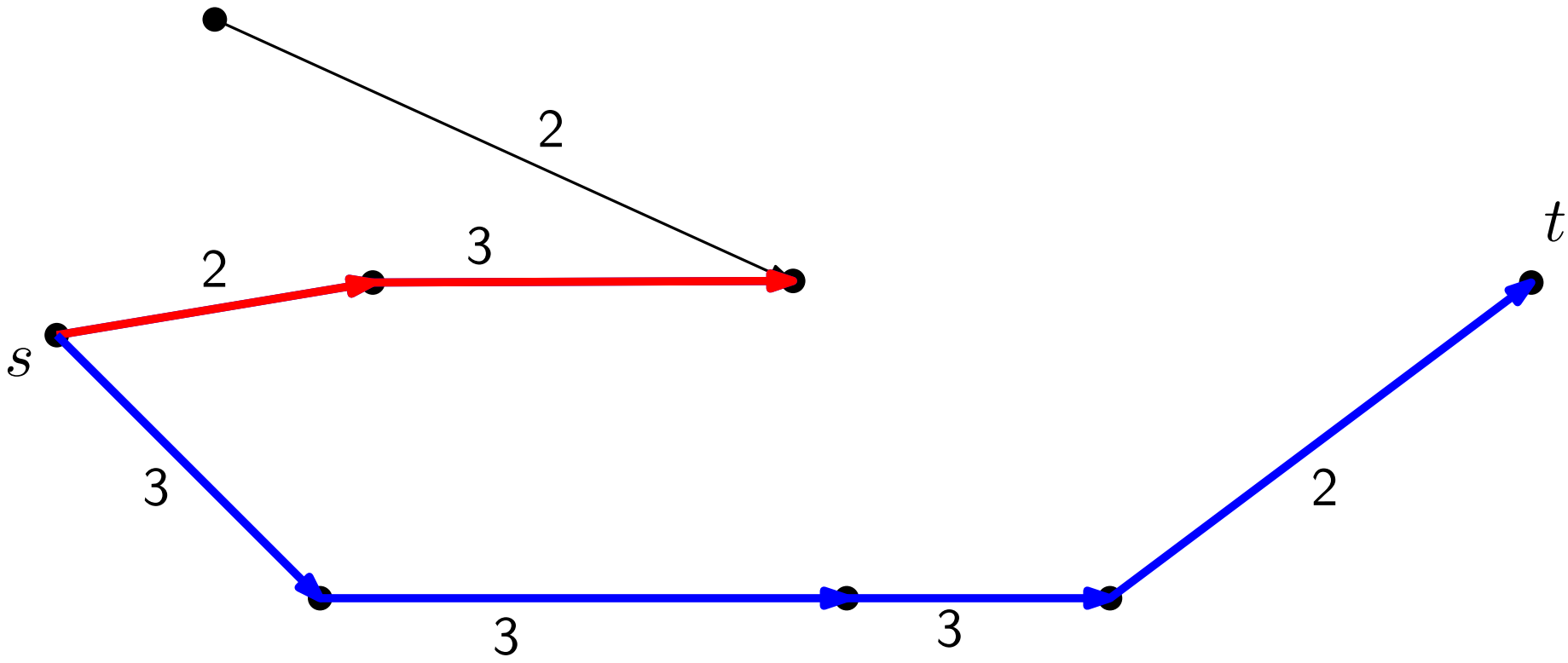
Find an s-t-path with depth-first search.



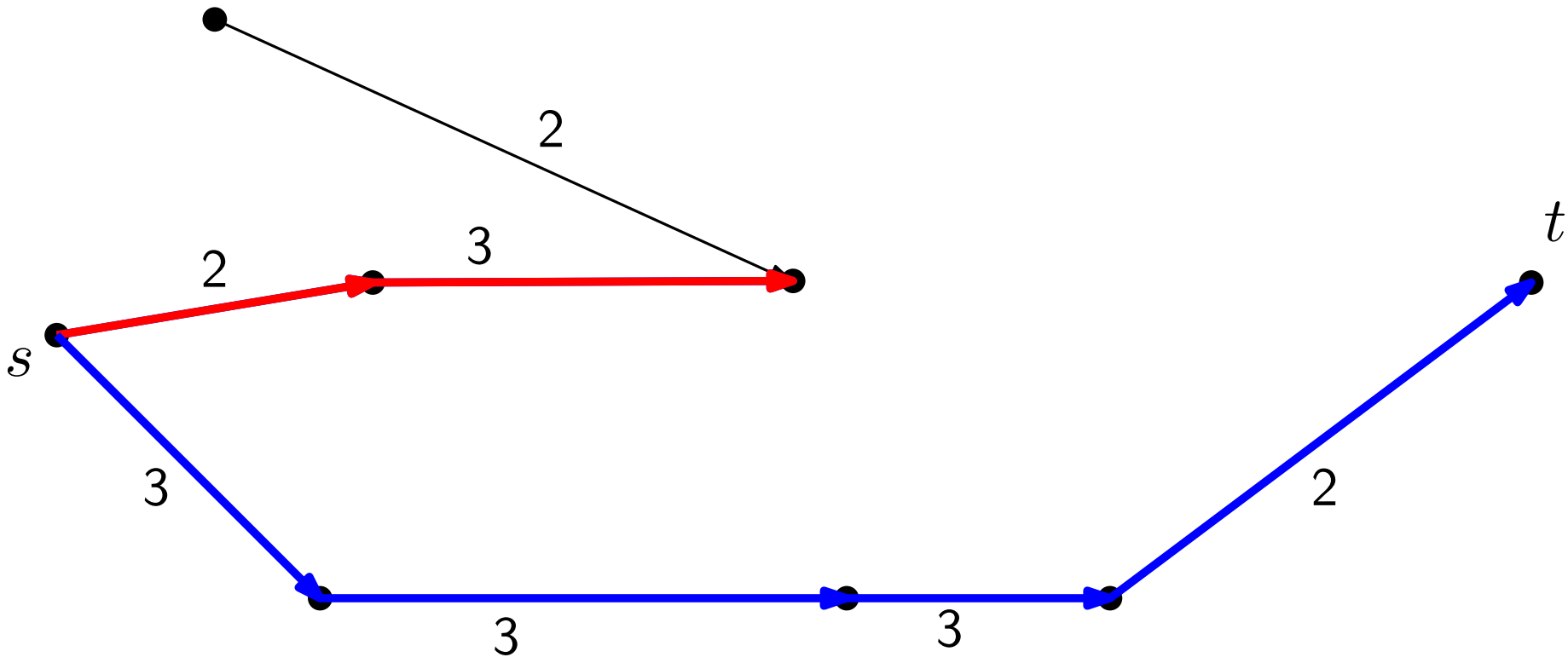
Find an s-t-path with depth-first search.



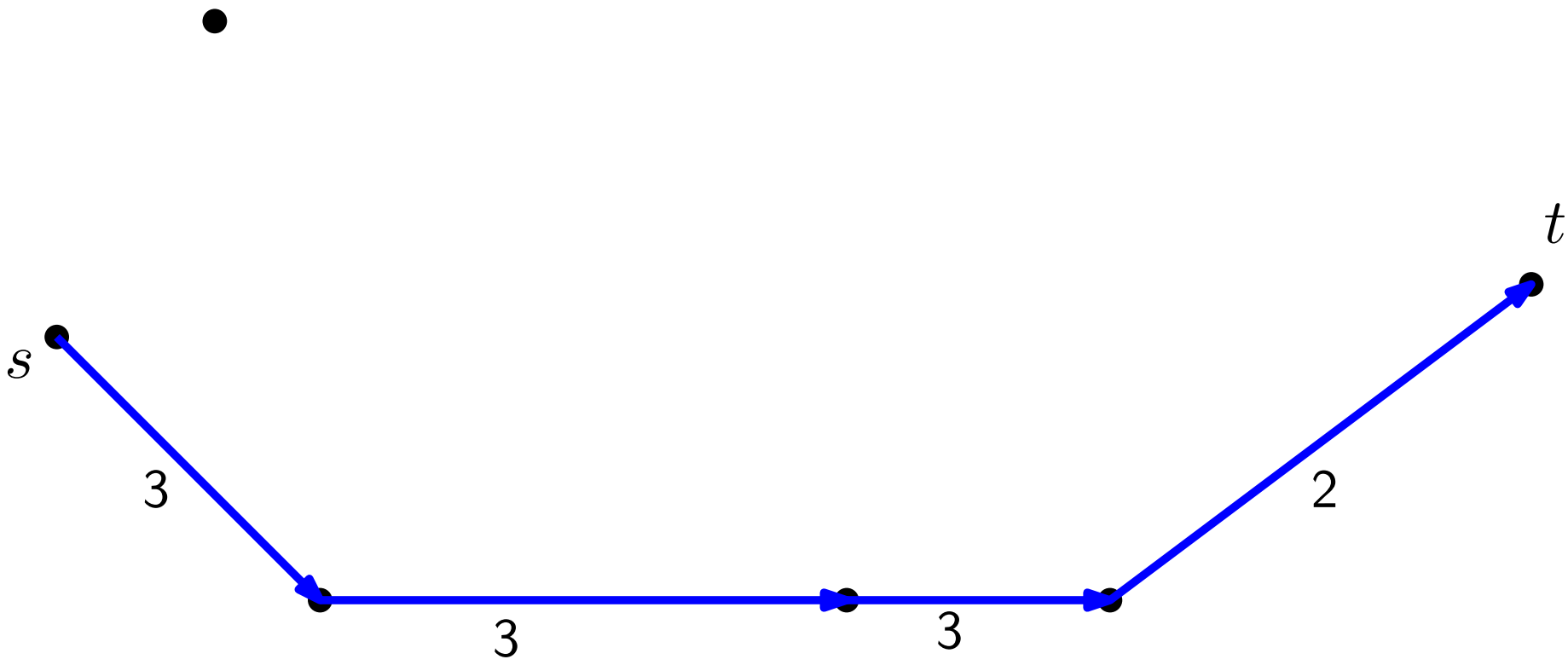
Find an s-t-path with depth-first search.



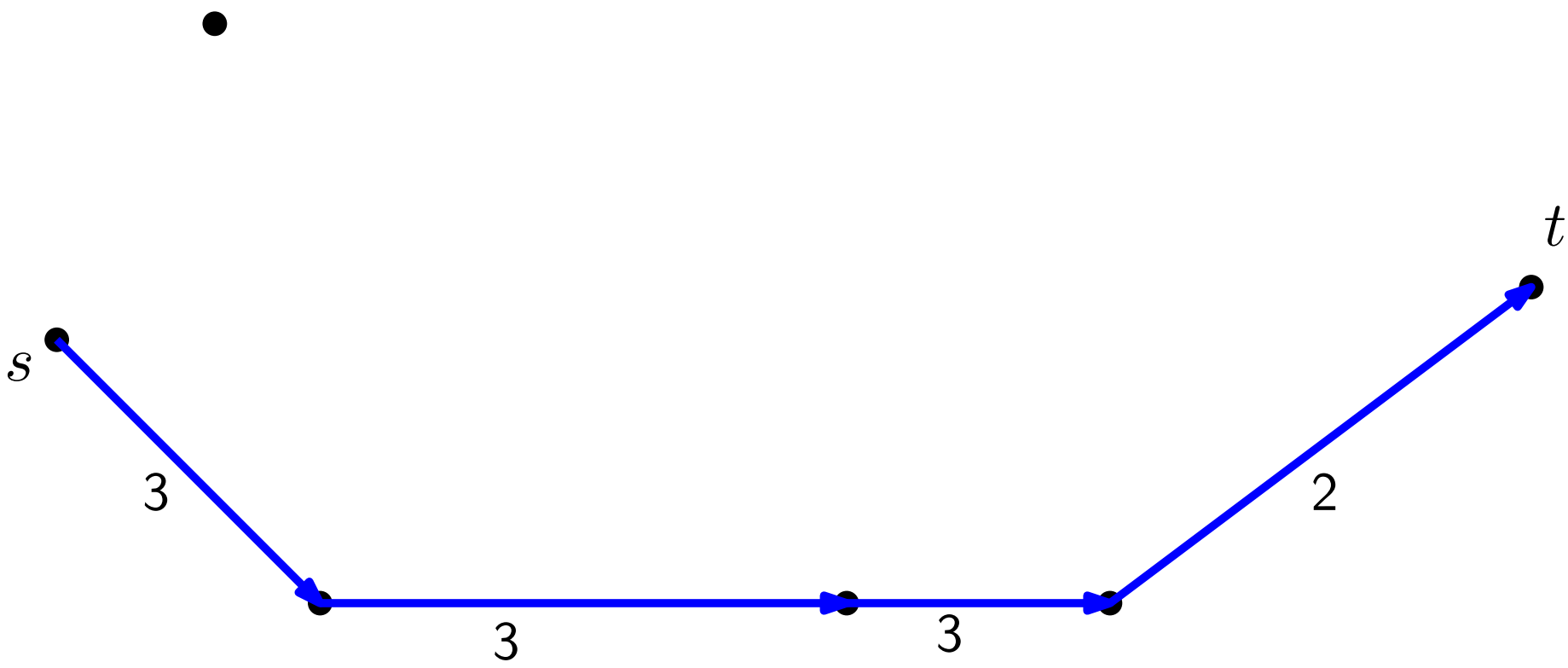
Find an s-t-path with depth-first search.



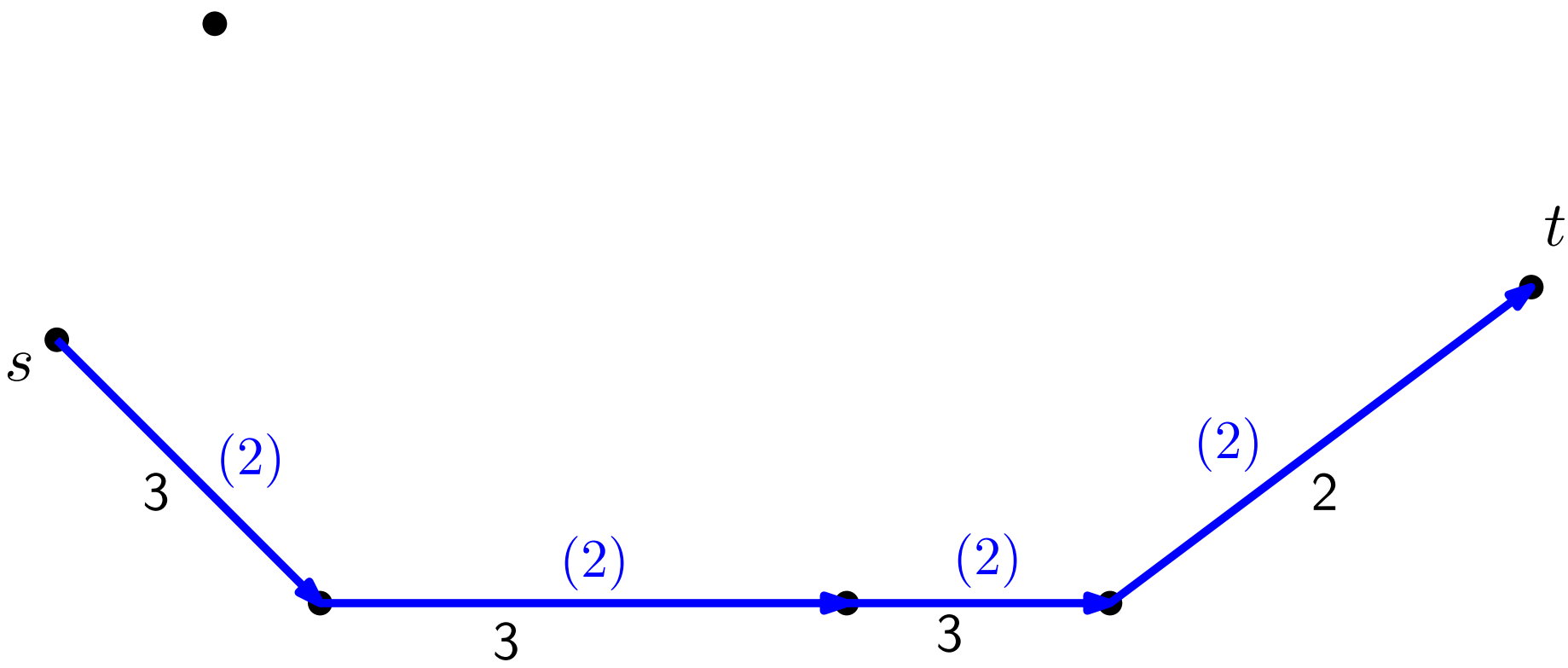
Find an s-t-path with depth-first search.
Delete dead ends.



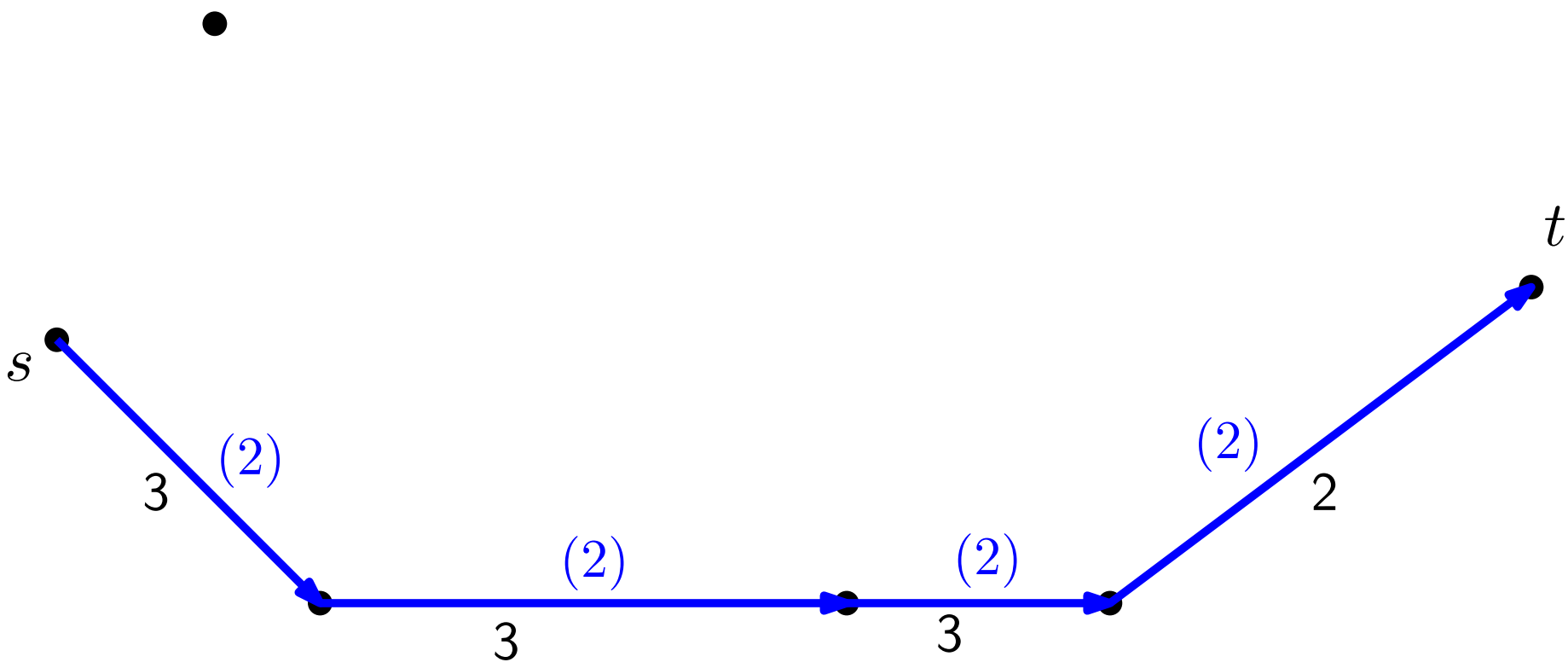
Find an s-t-path with depth-first search.
Delete dead ends.



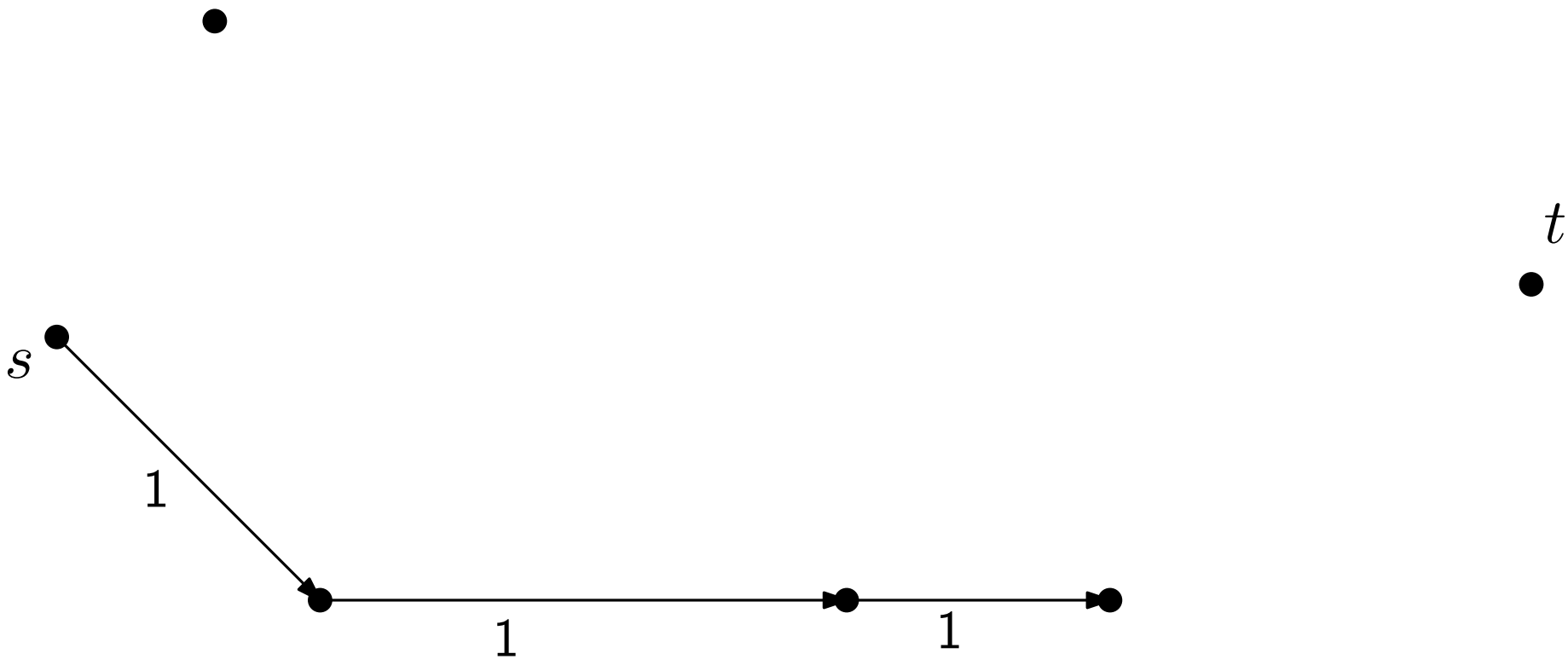
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.



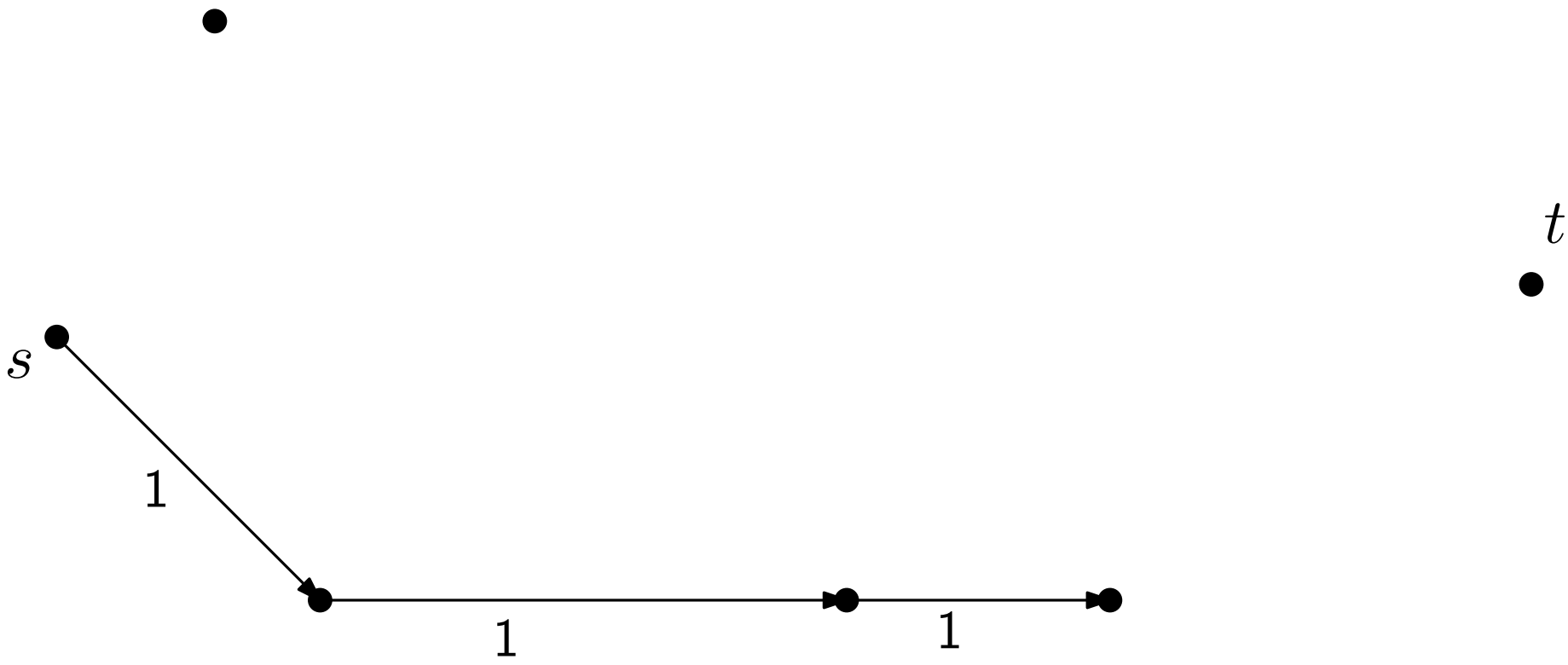
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.



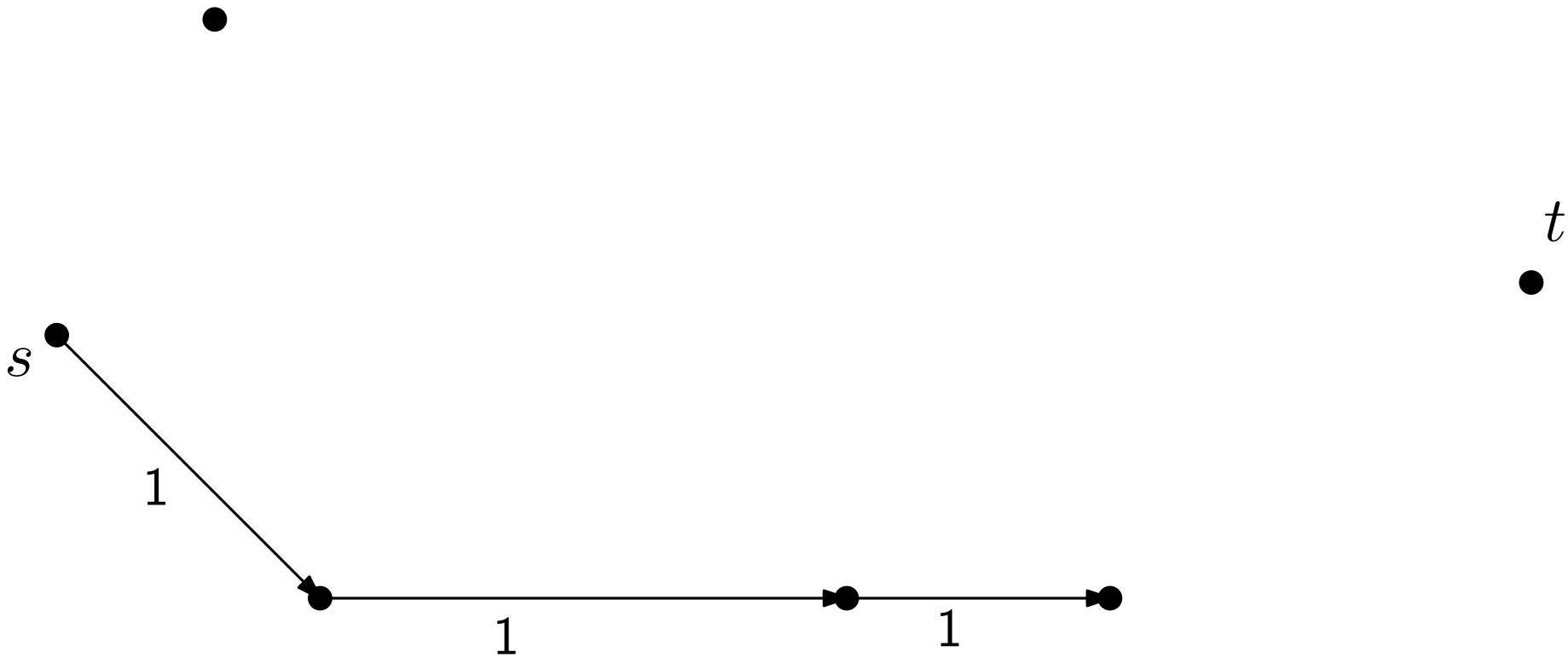
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.
Update capacities.



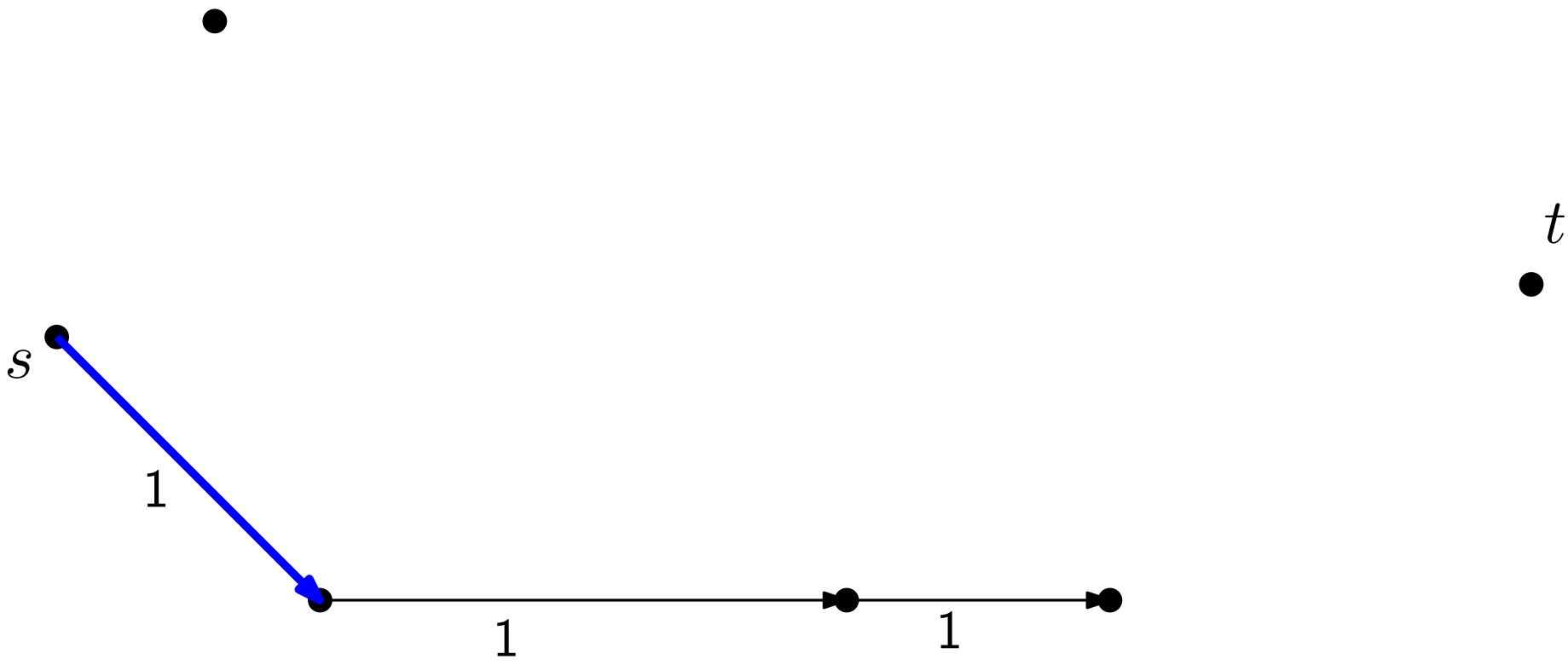
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.
Update capacities.



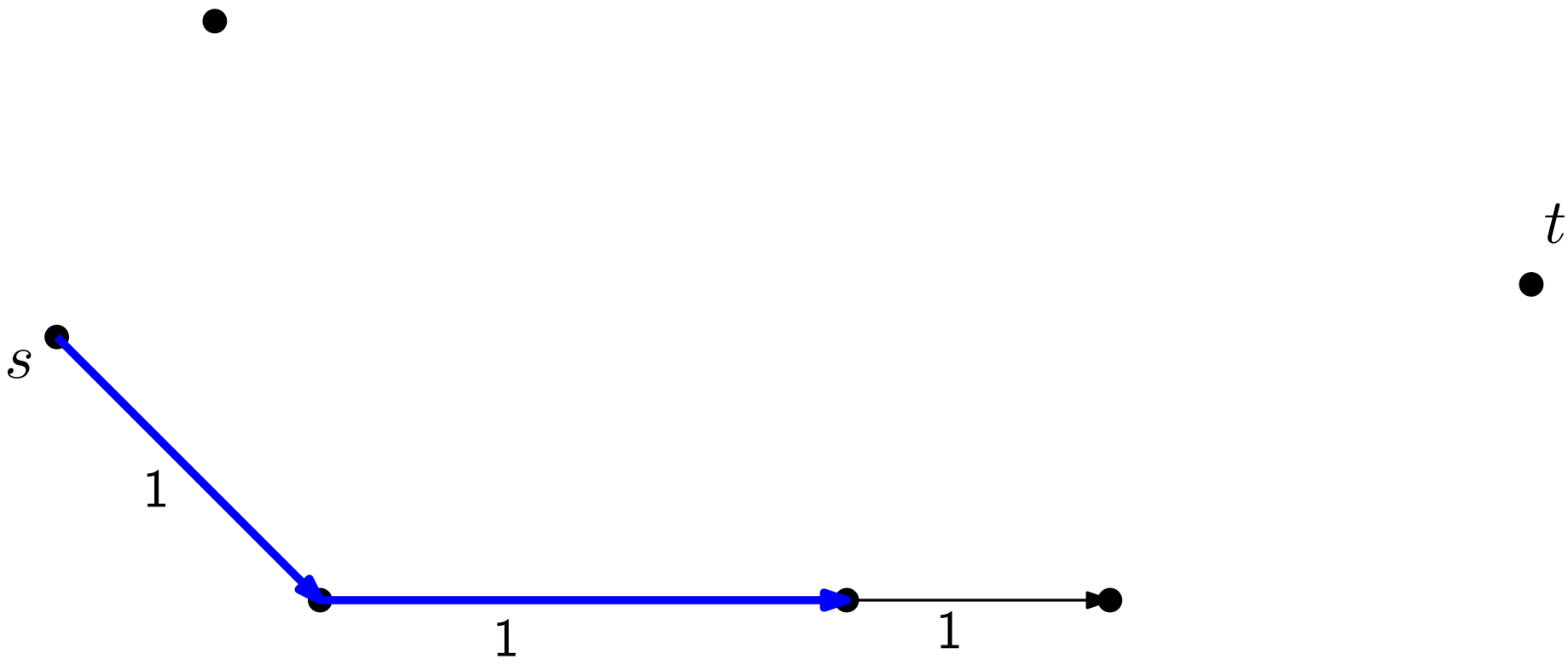
Find an s-t-path with depth-first search.
Delete dead ends.
Route as much flow as possible.
Update capacities. Repeat.



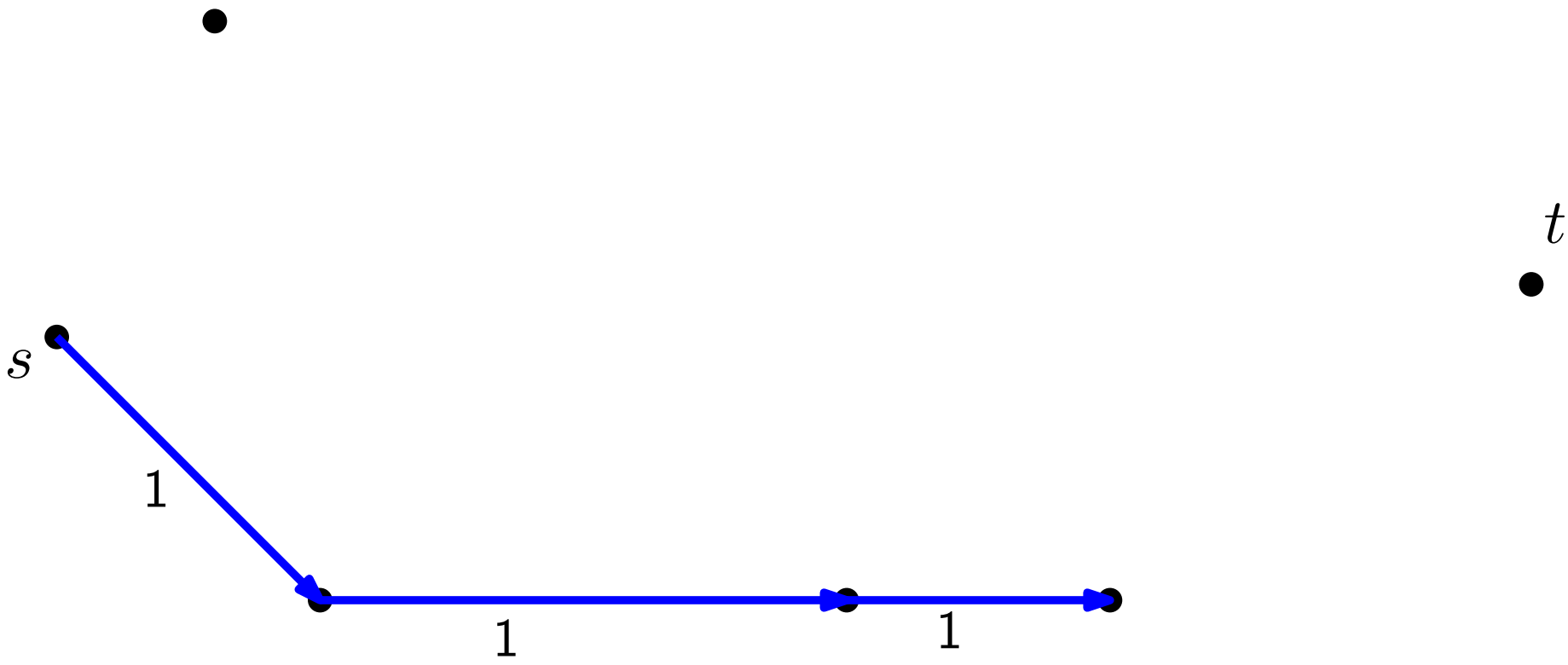
Find an s-t-path with depth-first search.



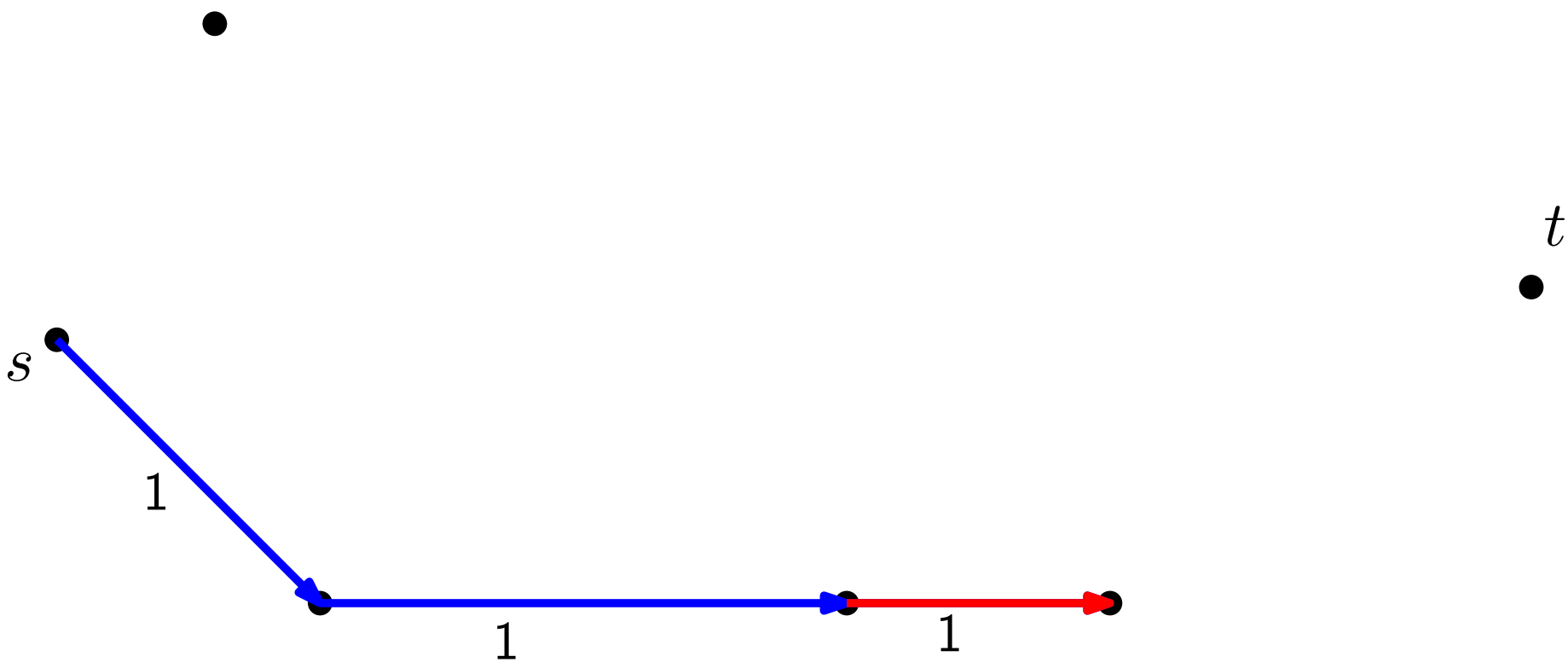
Find an s-t-path with depth-first search.



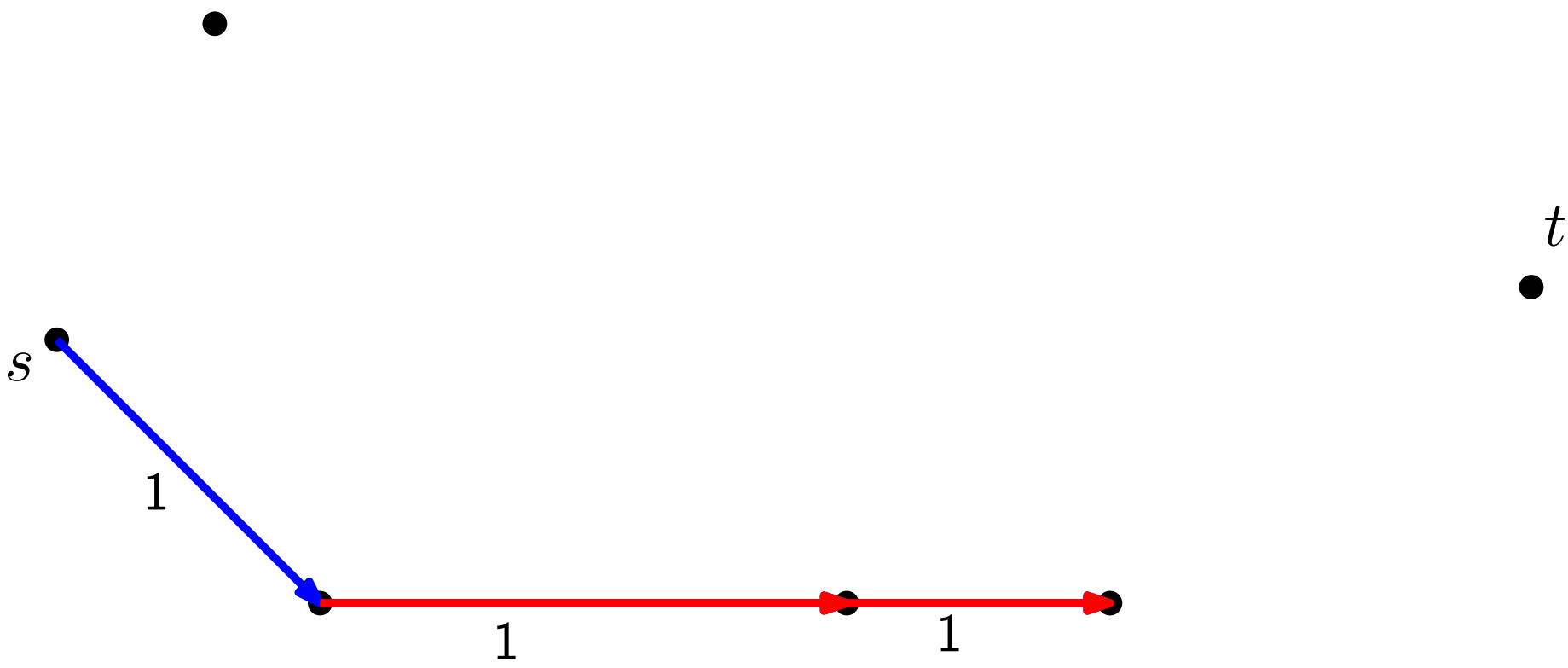
Find an s-t-path with depth-first search.



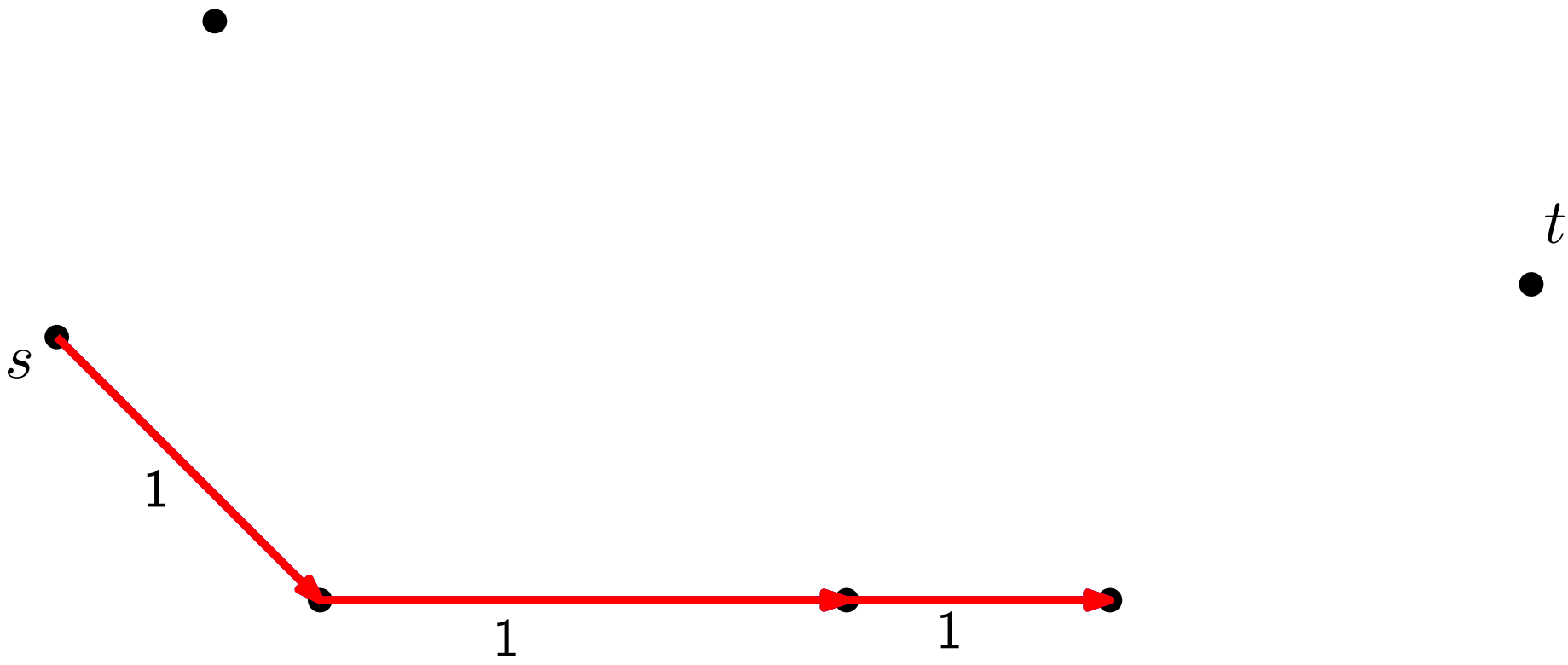
Find an s-t-path with depth-first search.



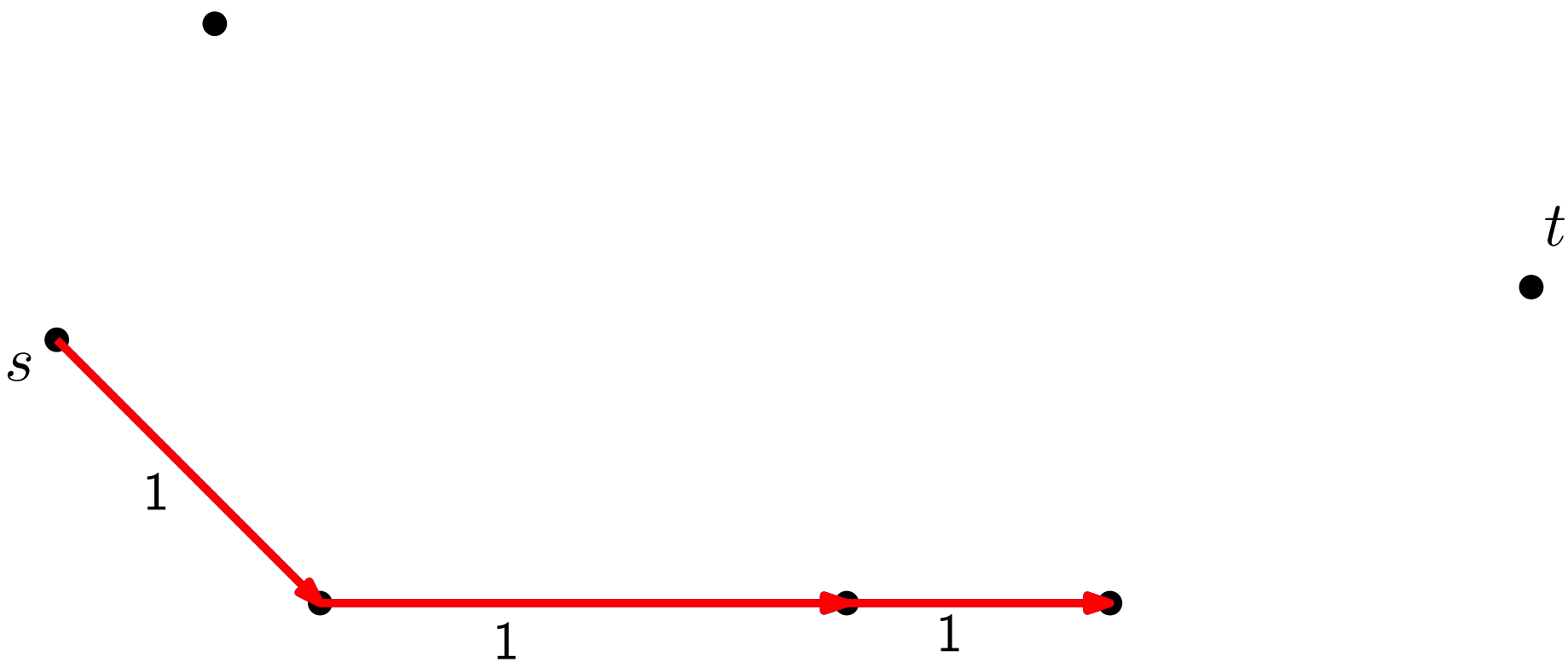
Find an s-t-path with depth-first search.



Find an s-t-path with depth-first search.

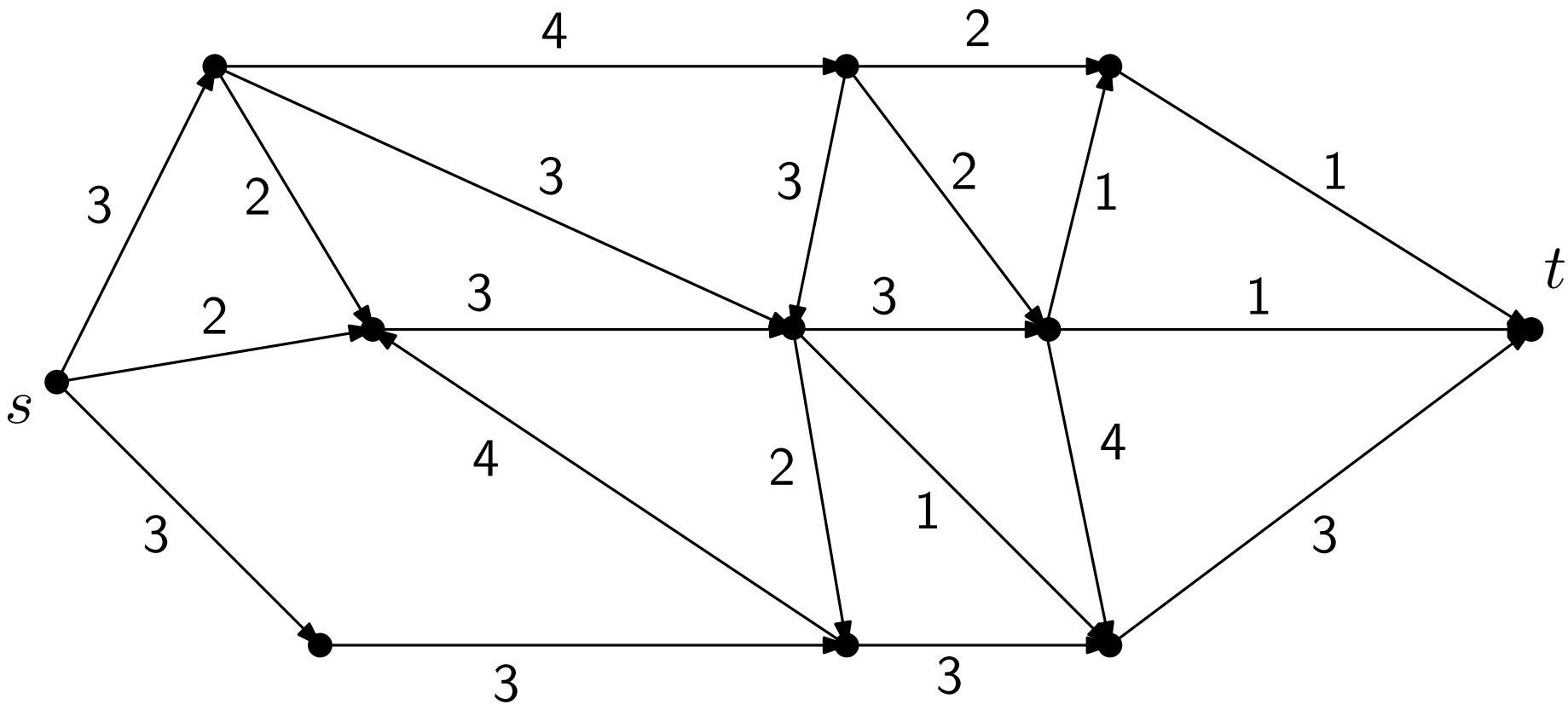


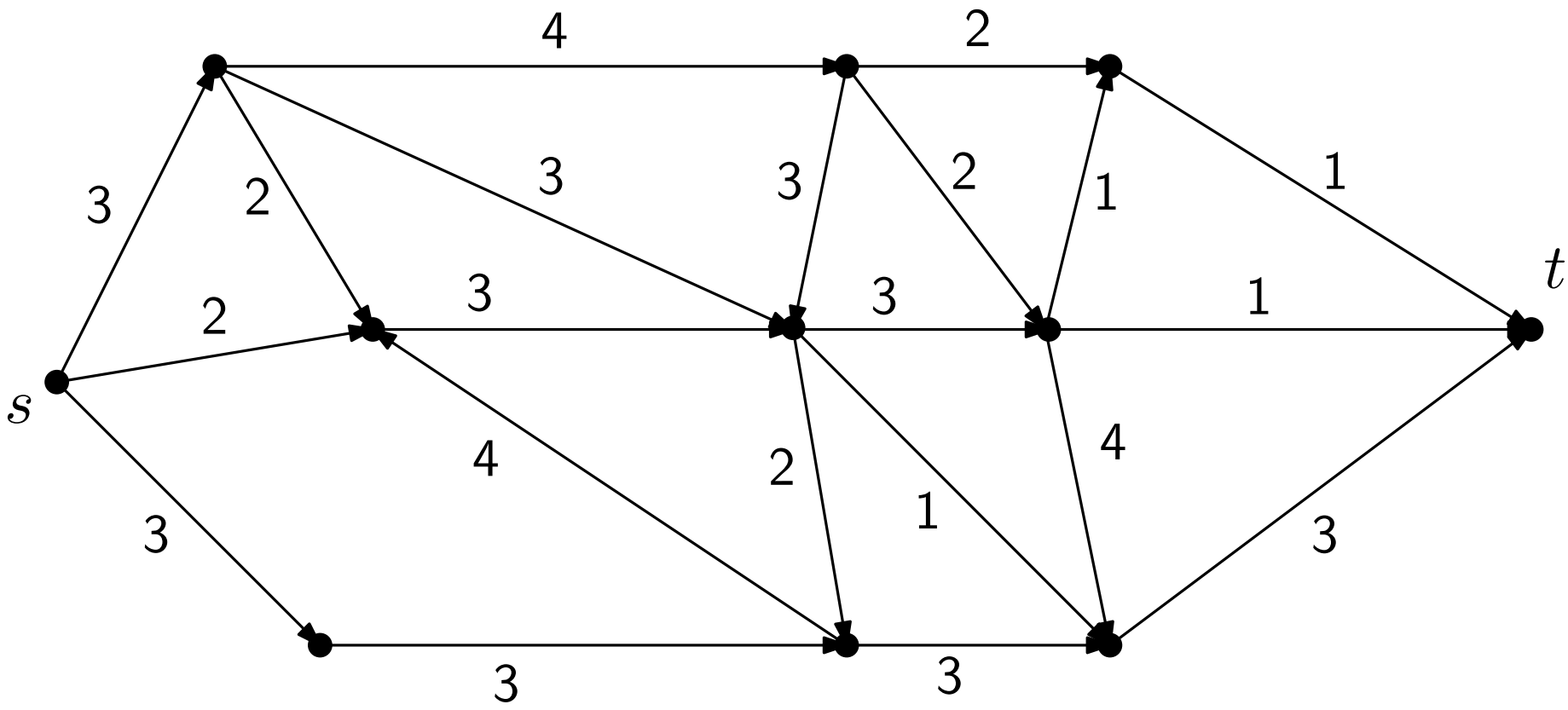
Find an s-t-path with depth-first search.



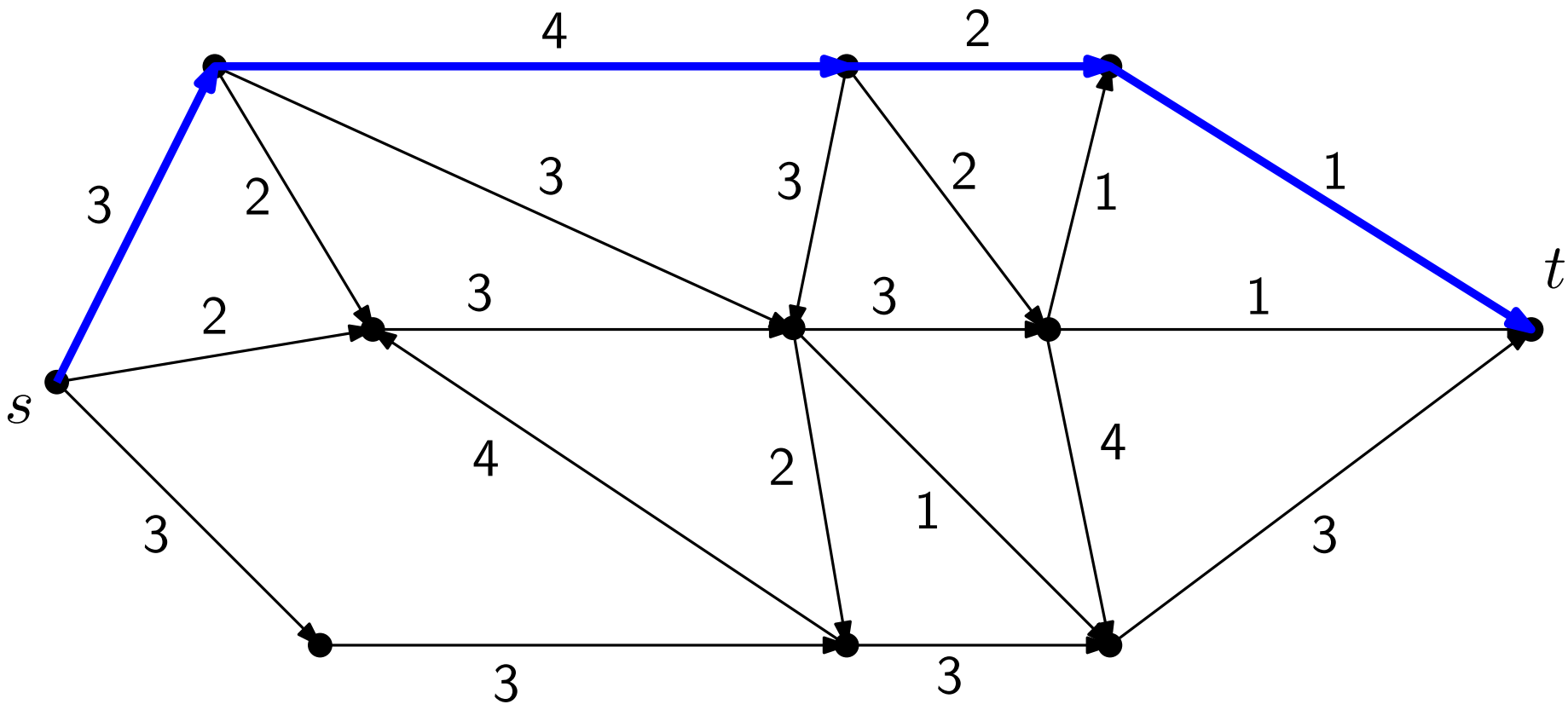
Find an s-t-path with depth-first search.
Delete dead ends.

Find an s-t-path with depth-first search.
Delete dead ends.
The source s has been deleted. The phase ends.

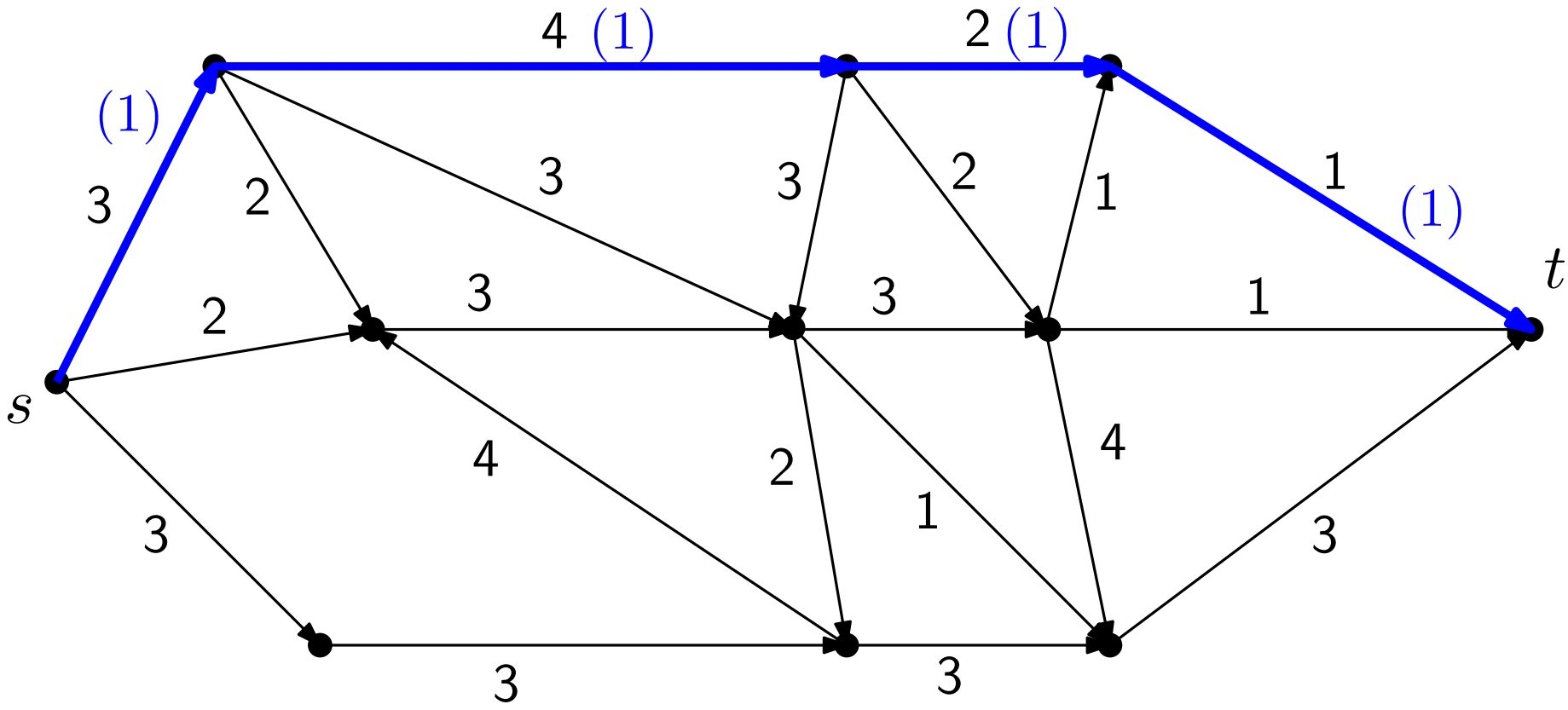

4
2
3
2
3
3
2
1
1
3
2
3
3
1
t
s
4
2
1
4
3
3
3



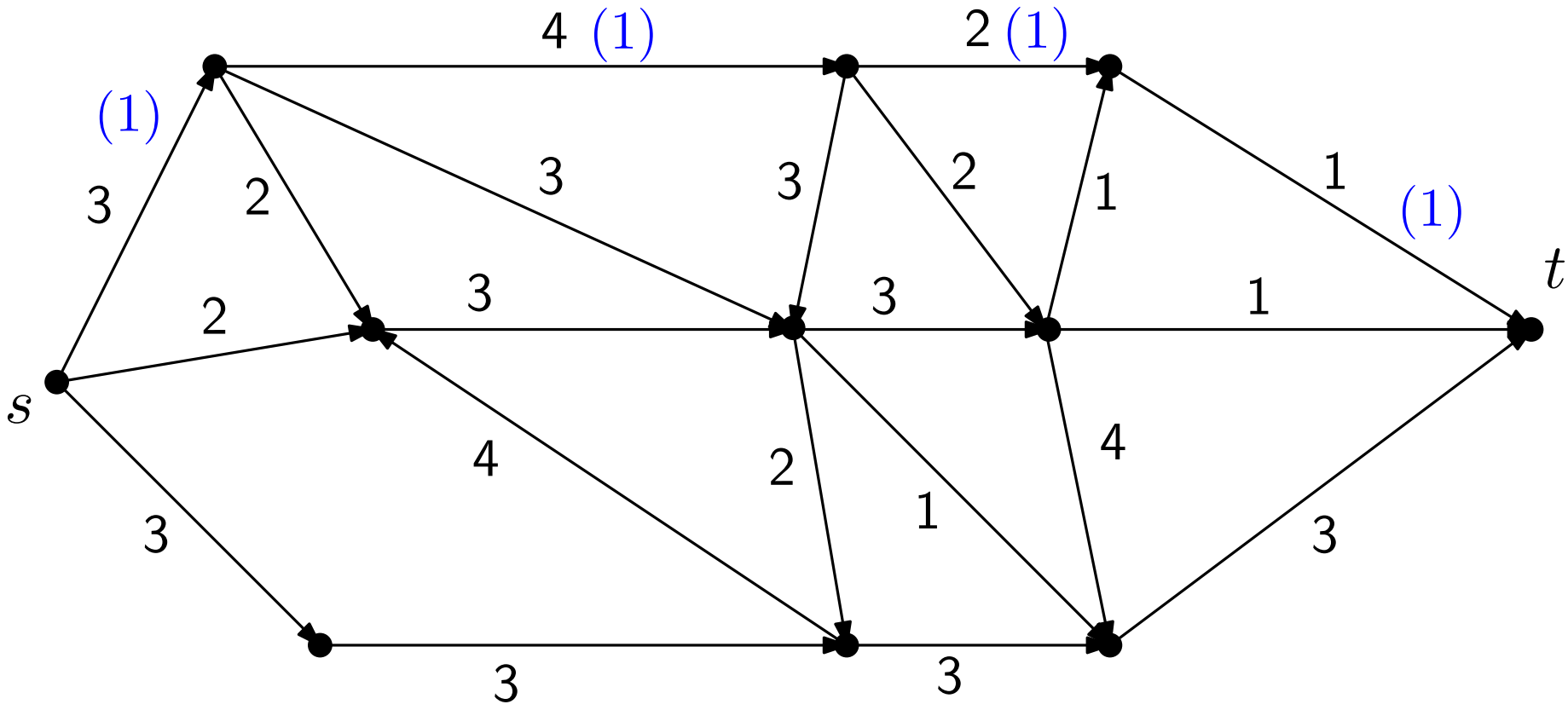
Collect the flow of this phase.



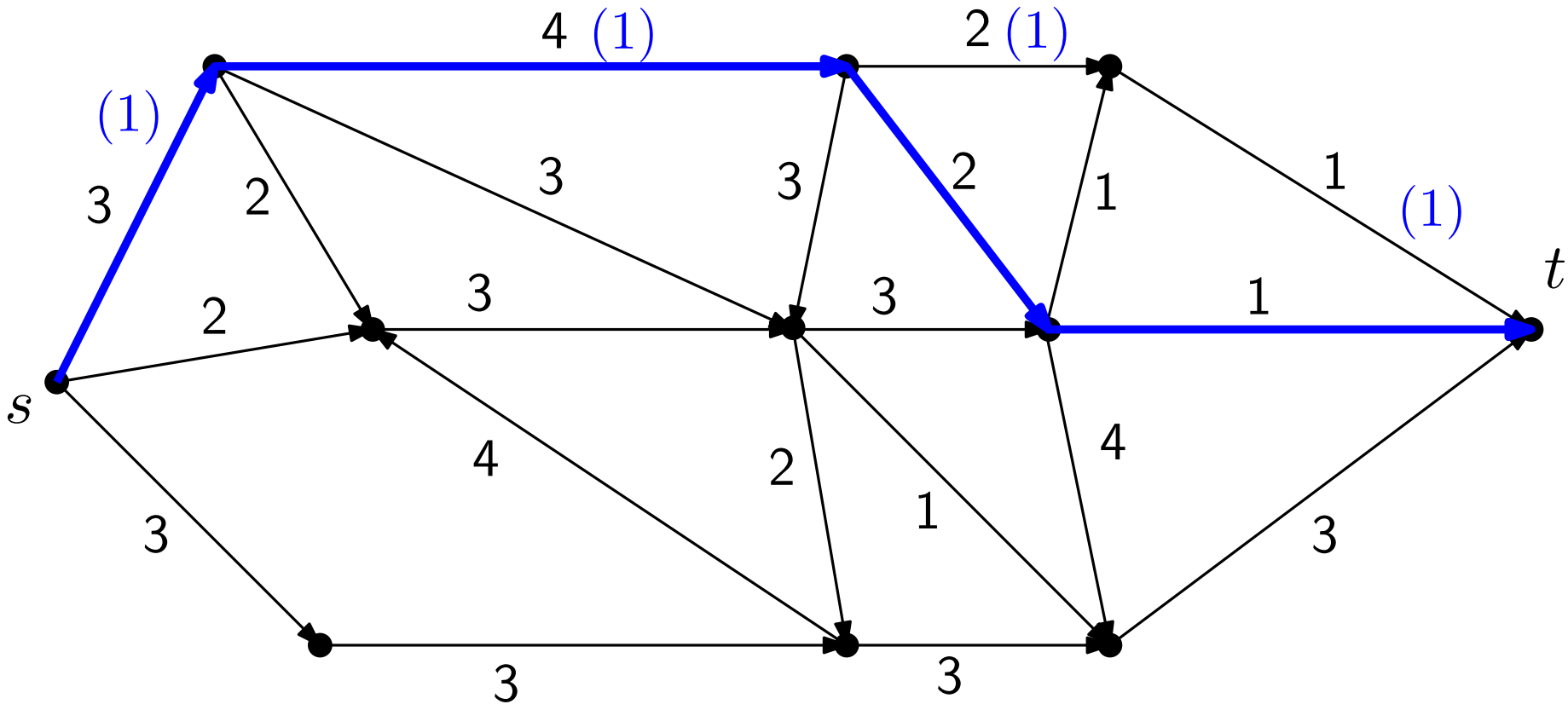
Collect the flow of this phase.



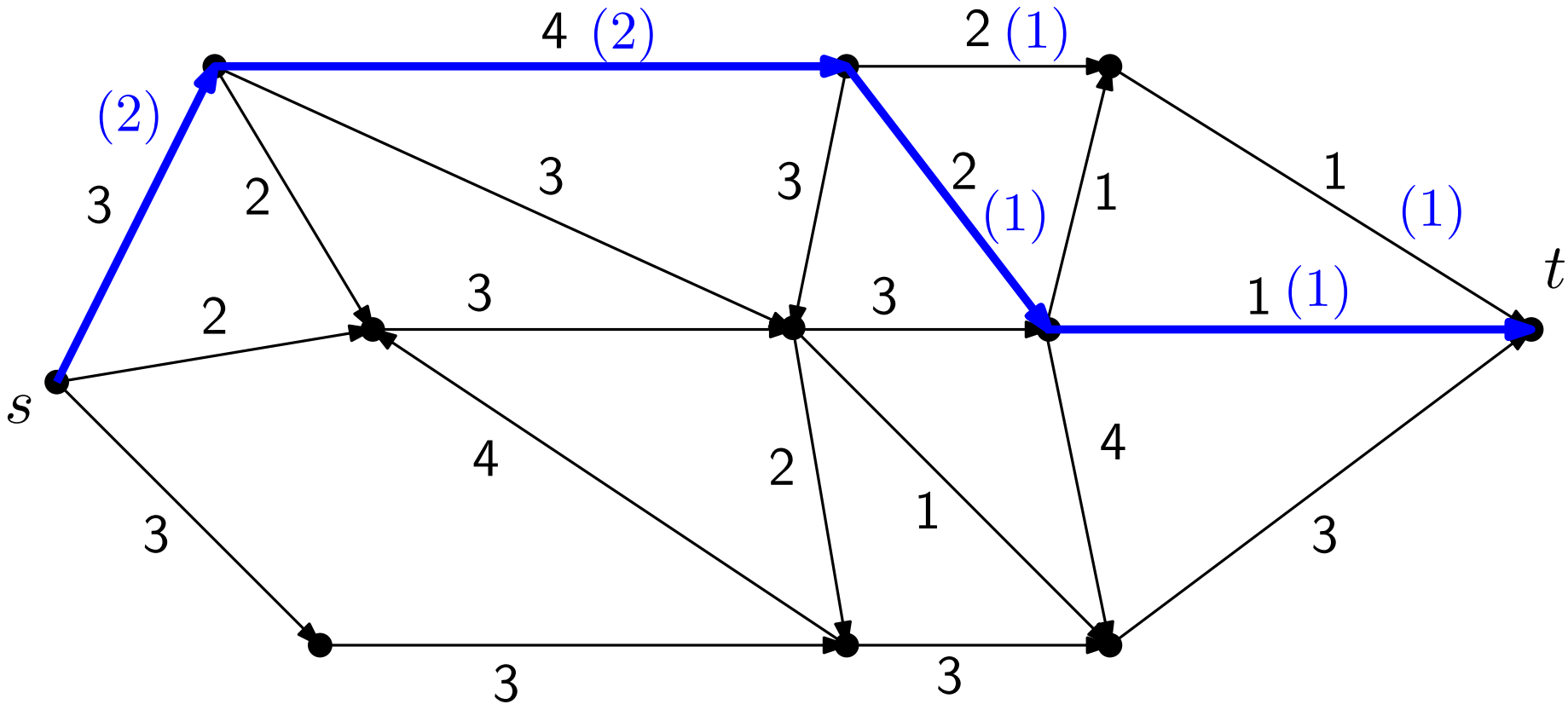
Collect the flow of this phase.



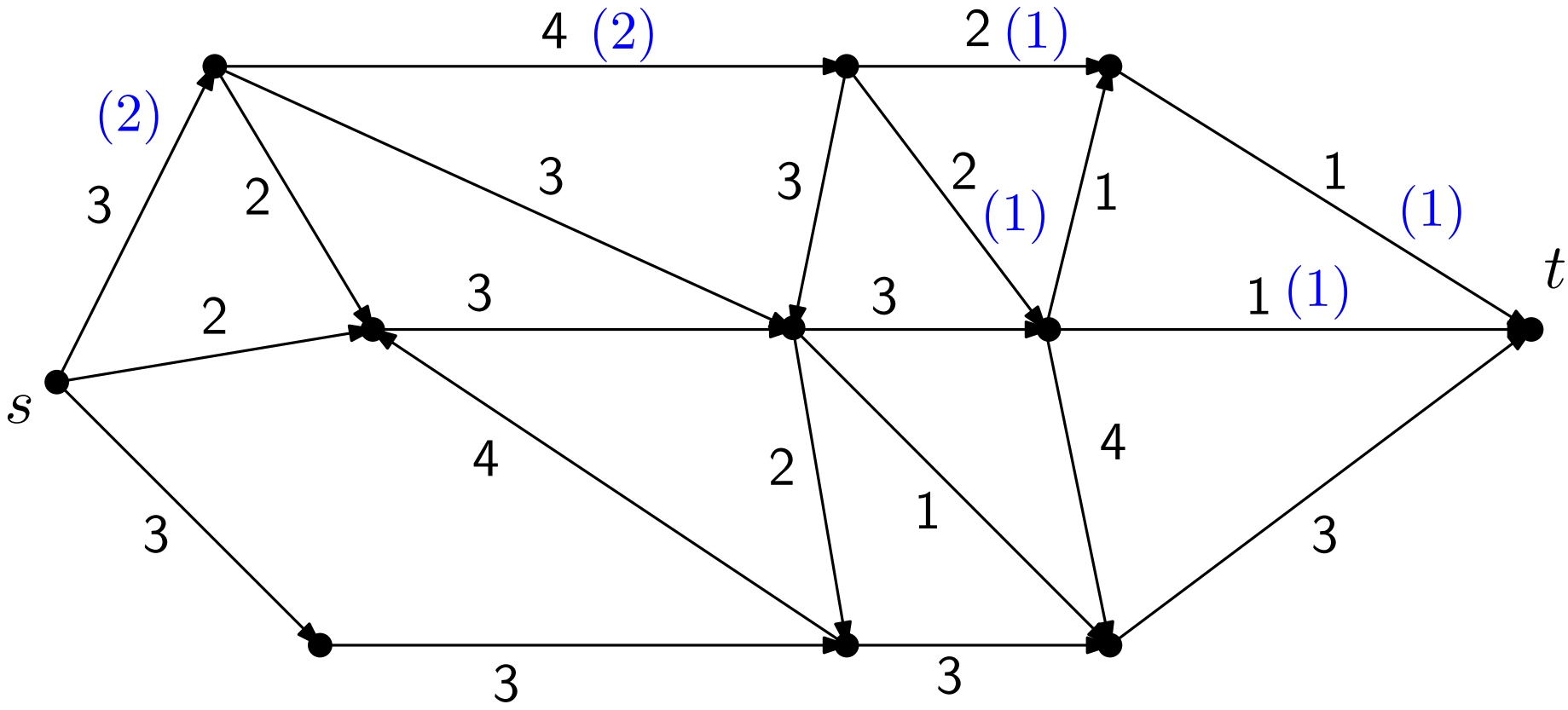
Collect the flow of this phase.



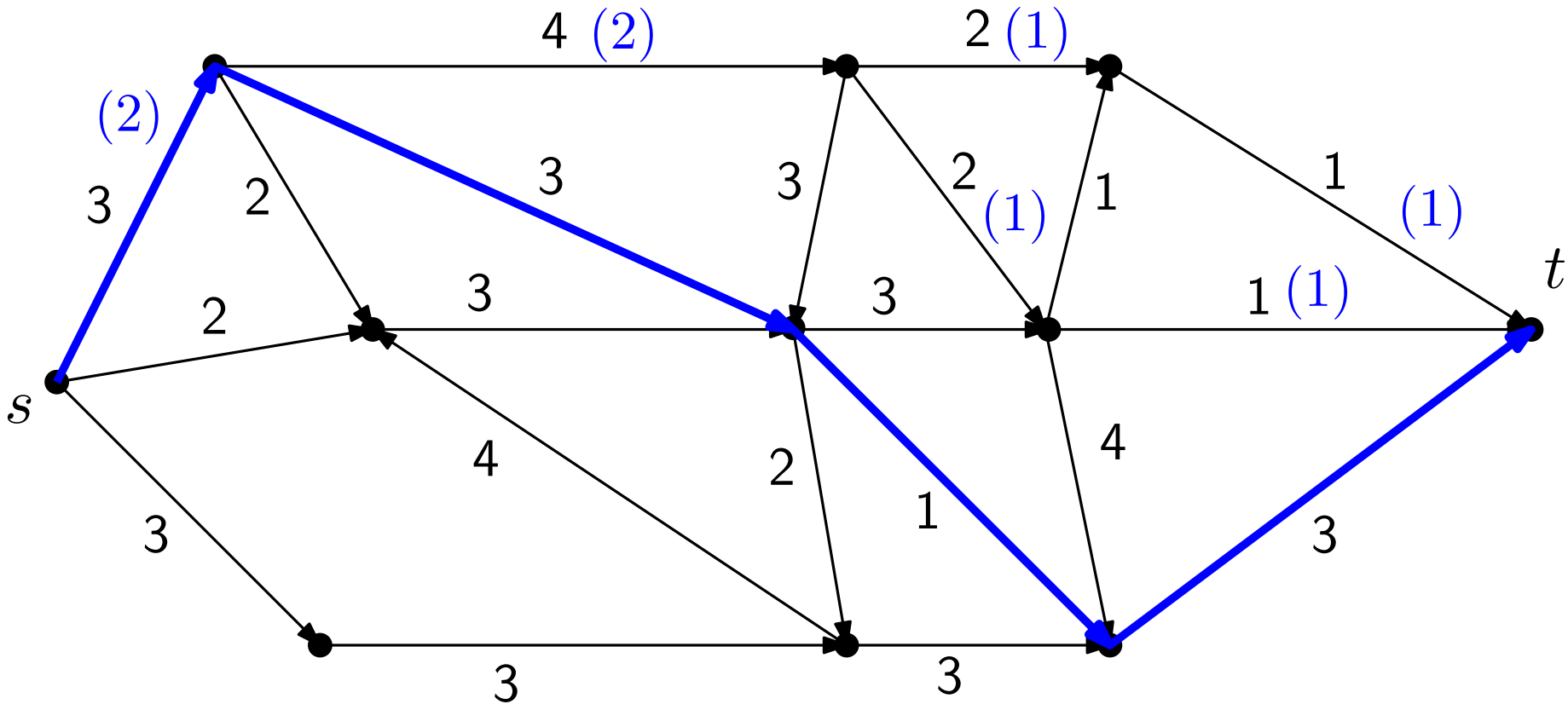
Collect the flow of this phase.



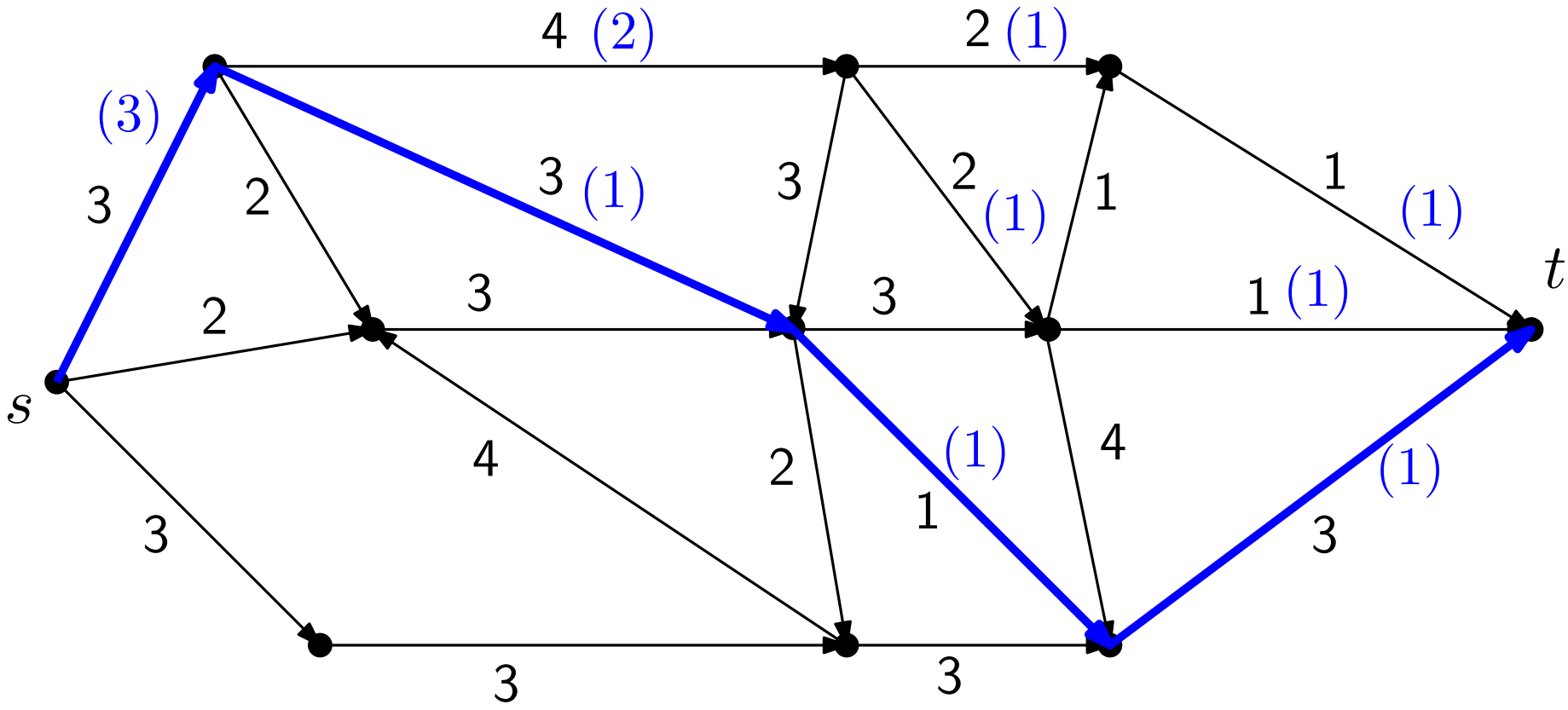
Collect the flow of this phase.



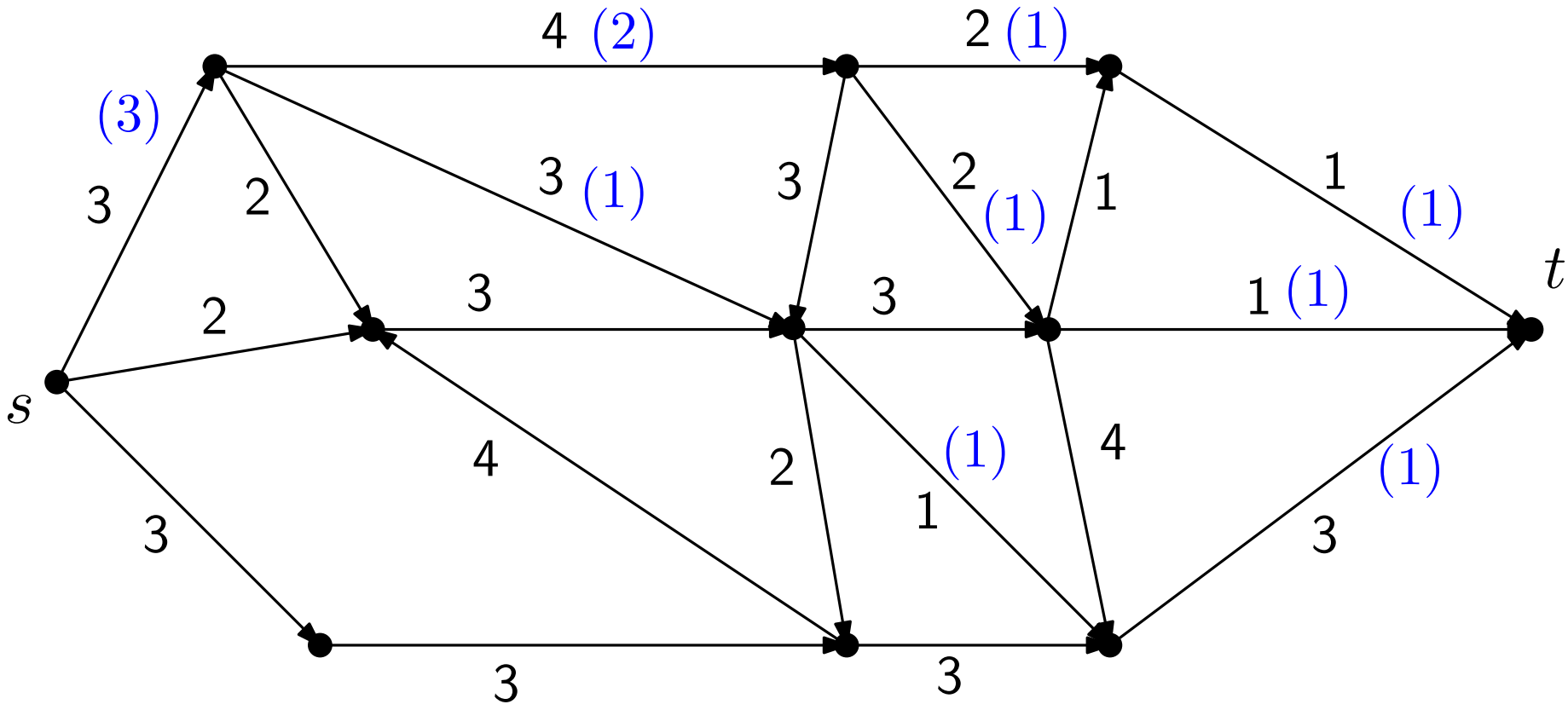
Collect the flow of this phase.



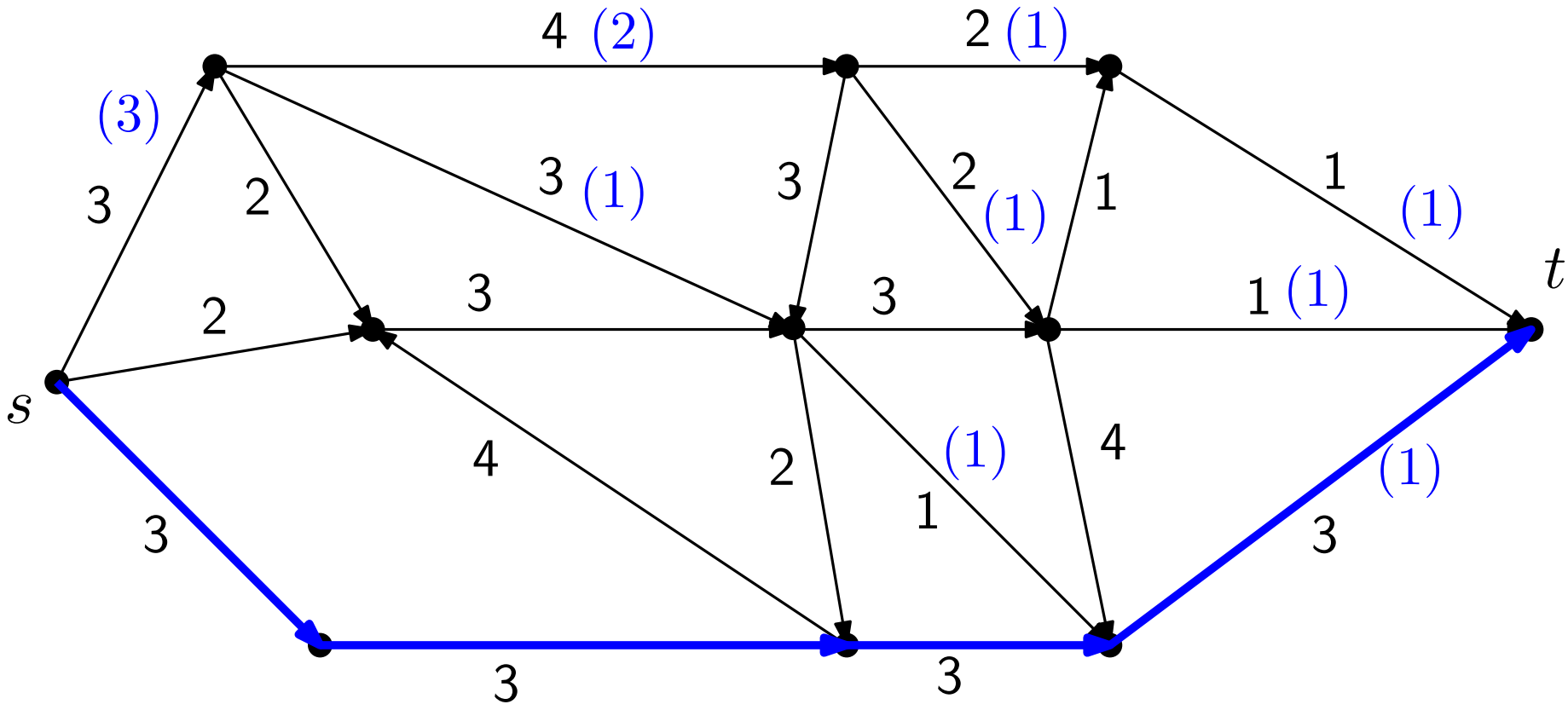
Collect the flow of this phase.



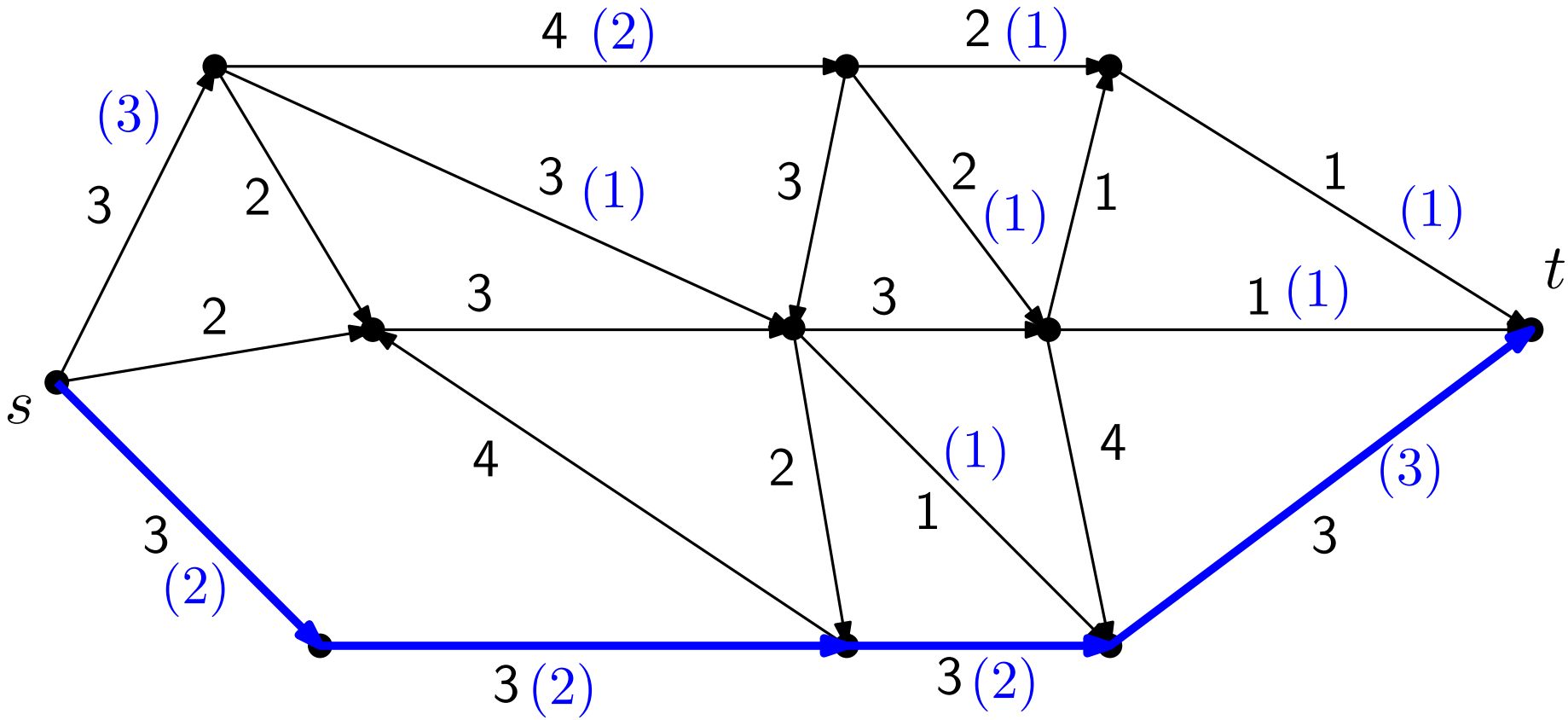
Collect the flow of this phase.



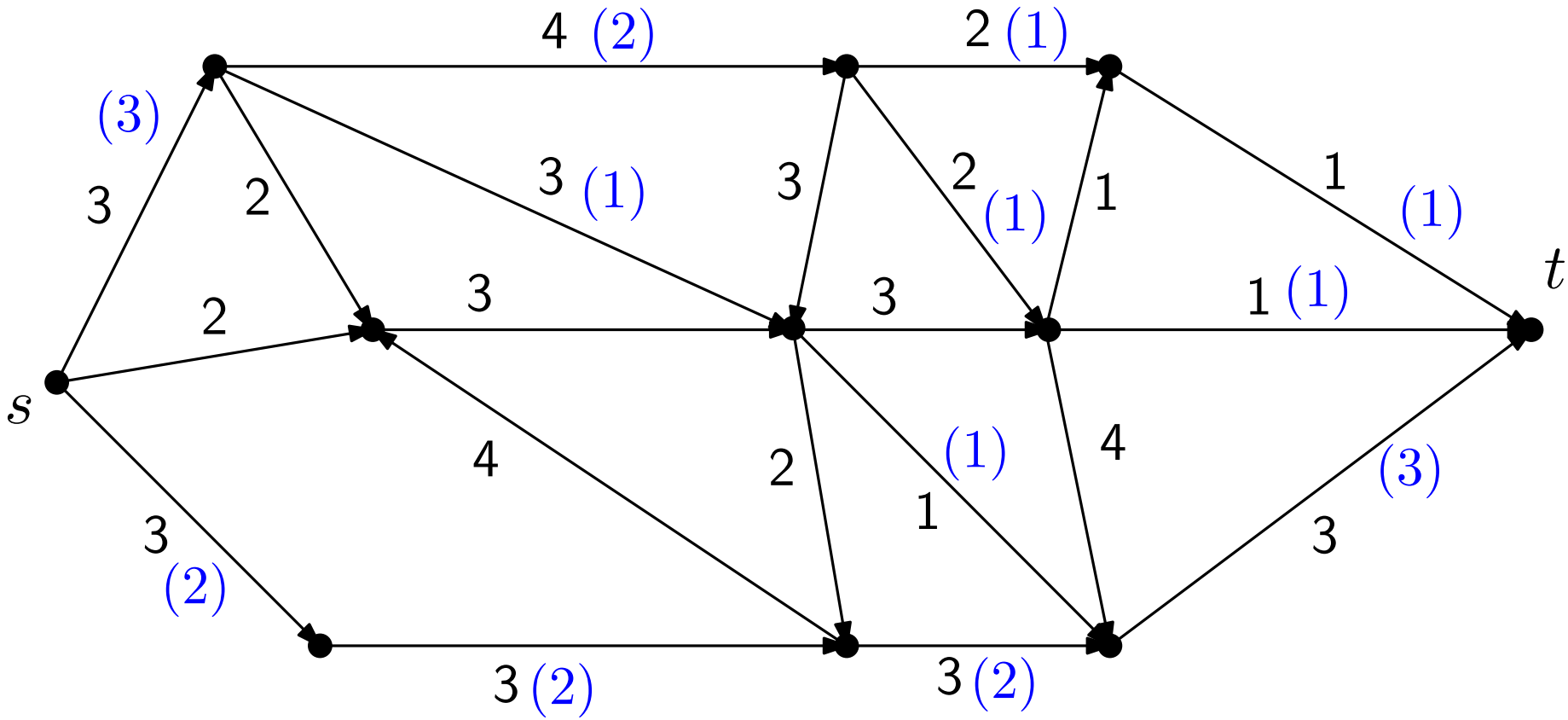
Collect the flow of this phase.



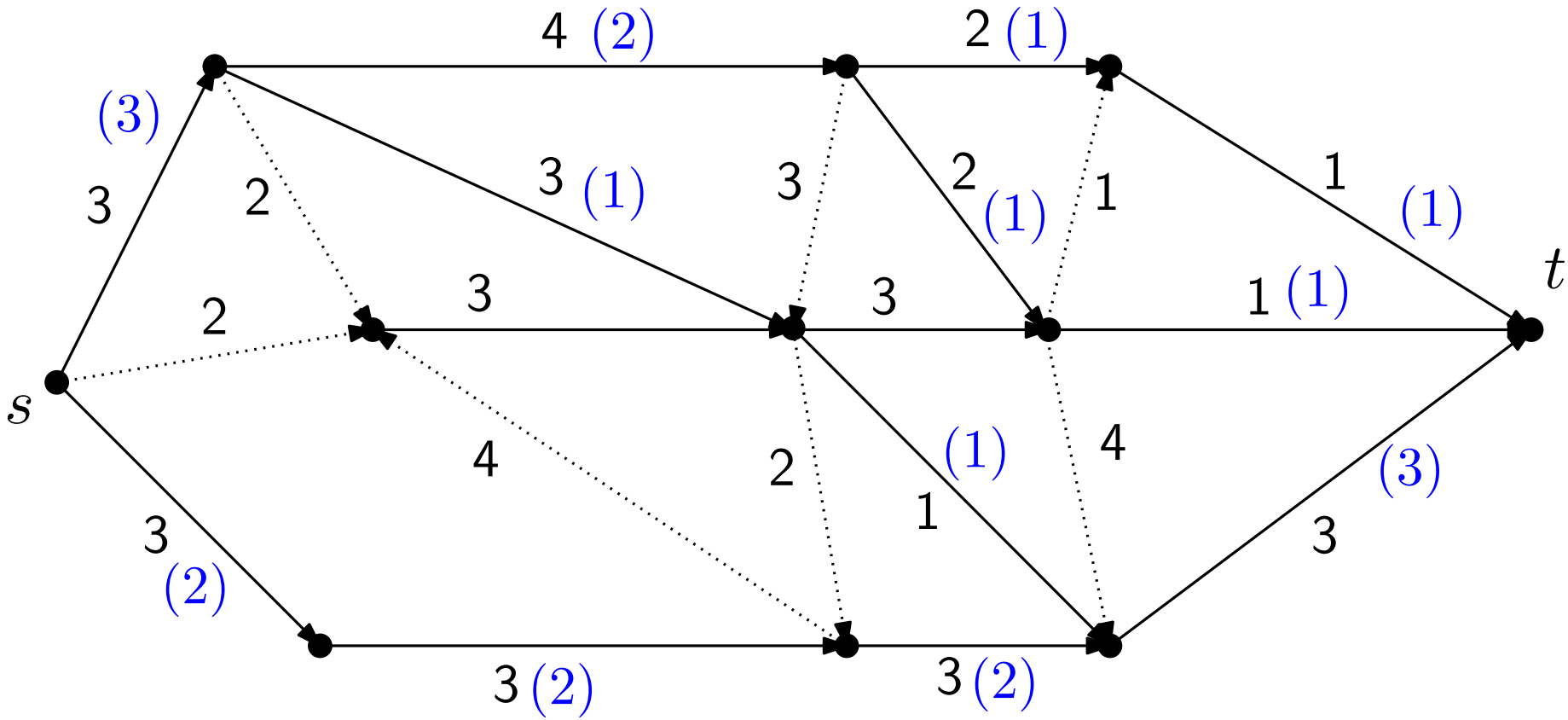
Collect the flow of this phase.



Collect the flow of this phase.

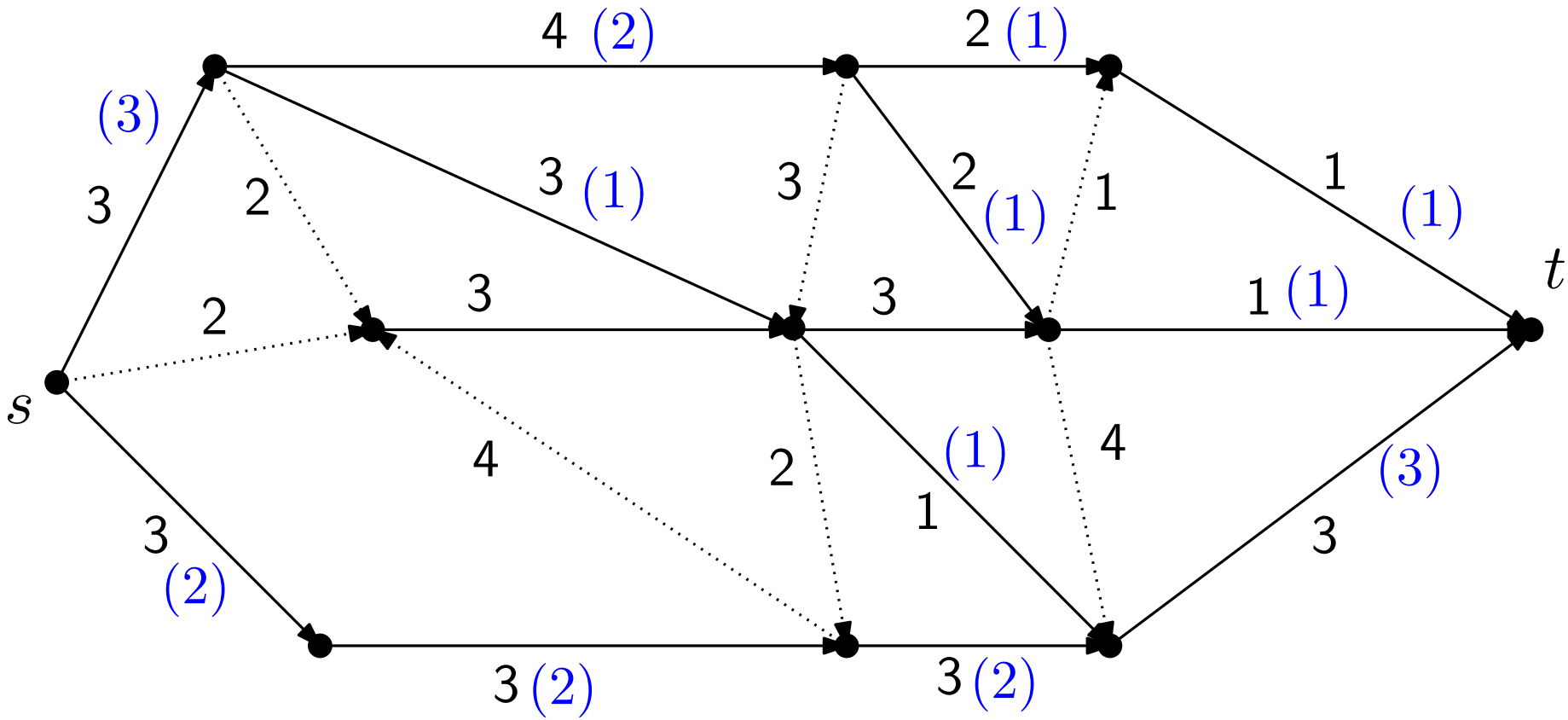


Collect the flow of this phase.



Collect the flow of this phase.

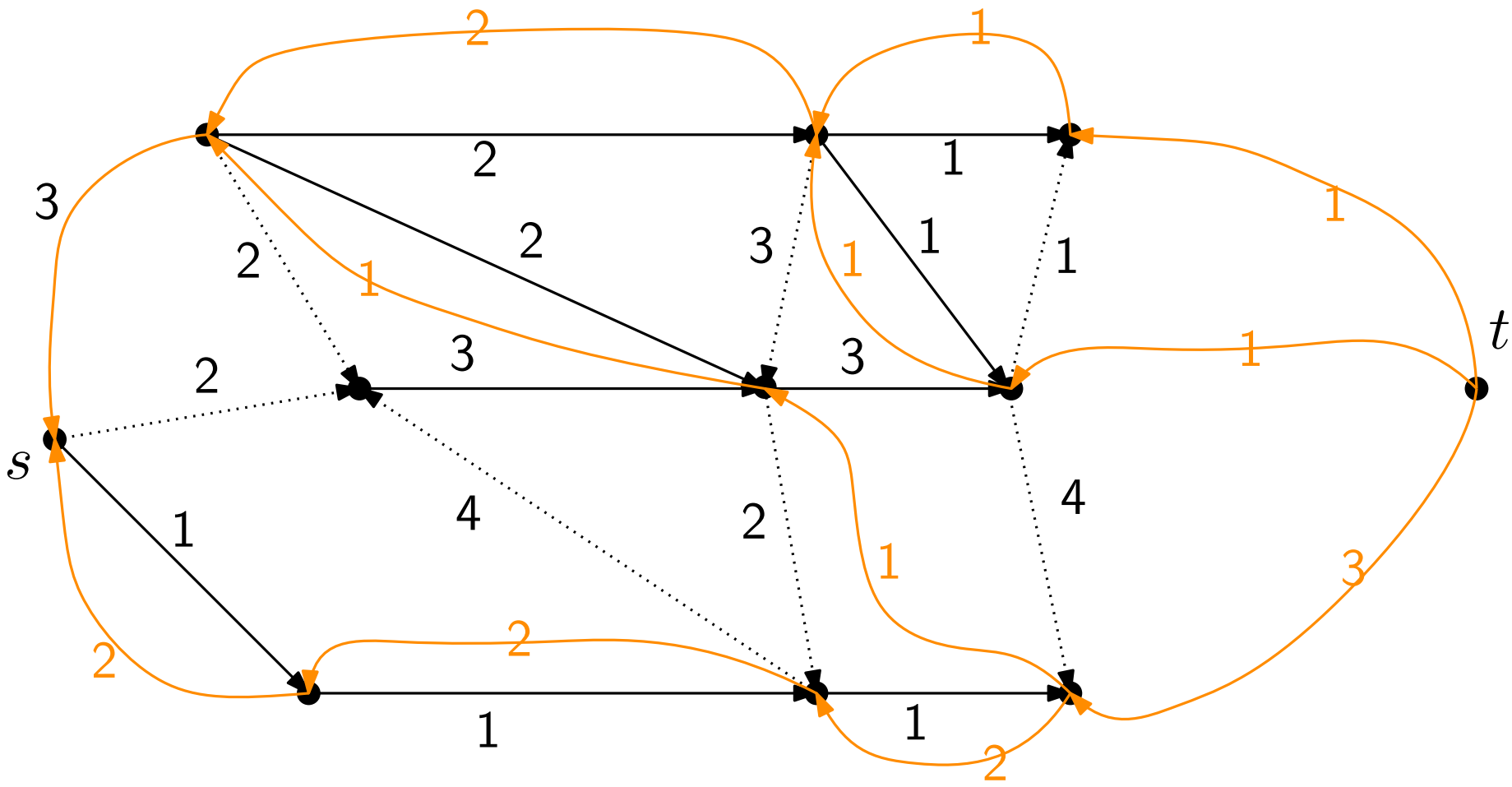
Observation: Only edges on a shortest s-t-path carry flow.



Collect the flow of this phase.

Observation: Only edges on a shortest s-t-path carry flow.

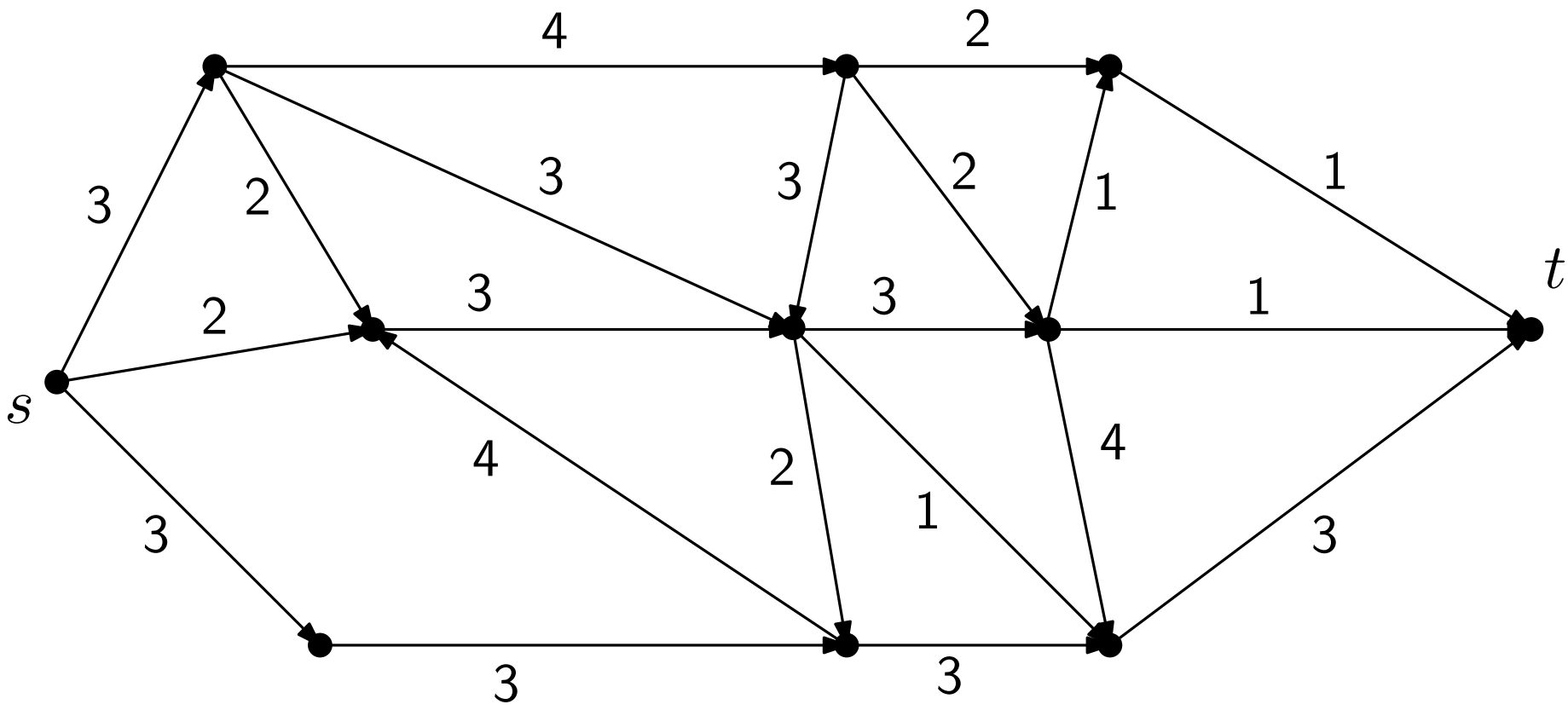
Build residual network (and introduce back edges)!

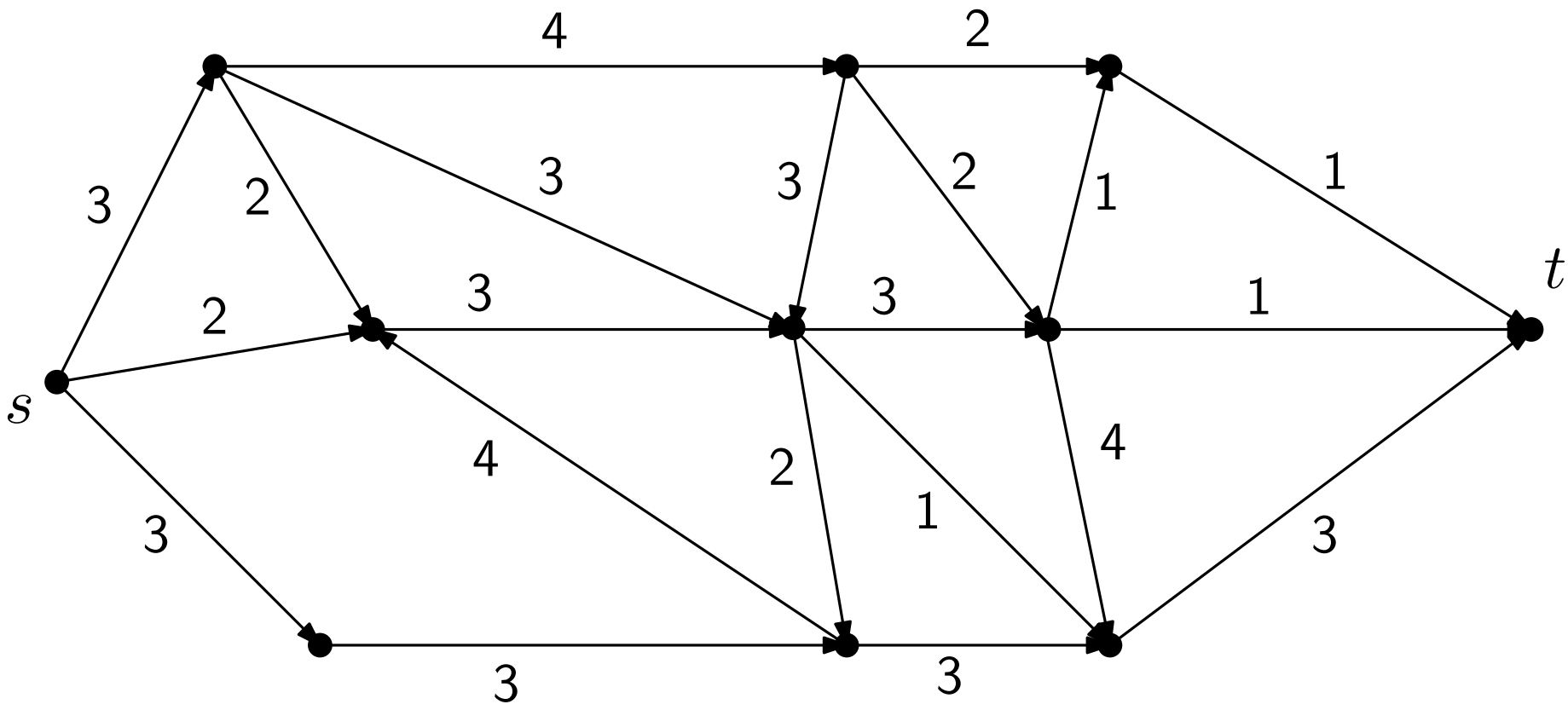


Collect the flow of this phase.

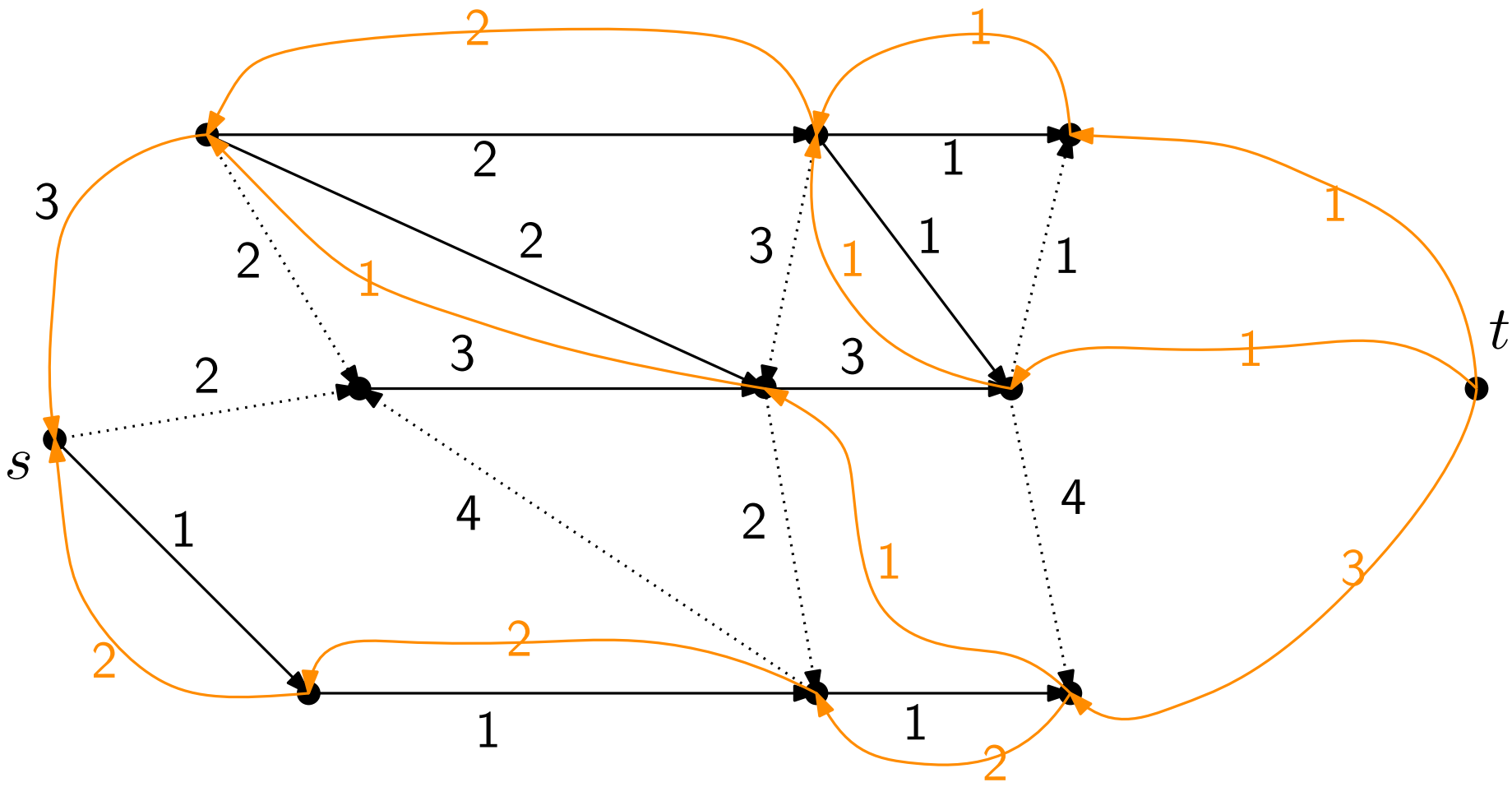
Observation: Only edges on a shortest s-t-path carry flow.

Build residual network (and introduce back edges)!

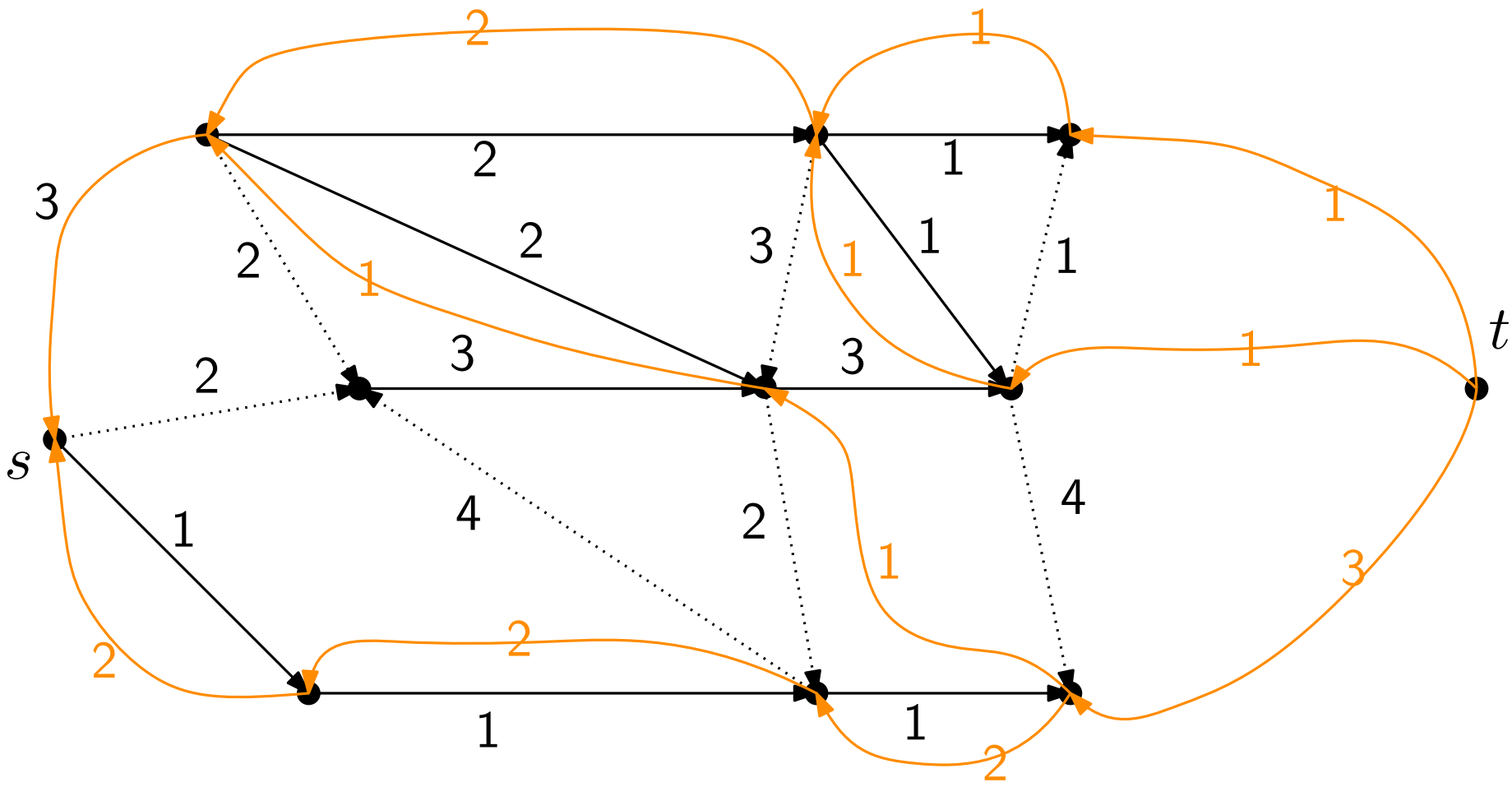

4
2
3
2
3
3
2
1
1
3
2
3
3
1
t
s
4
2
1
4
3
3
3



dist$(s, t) = 4$ in the original network.

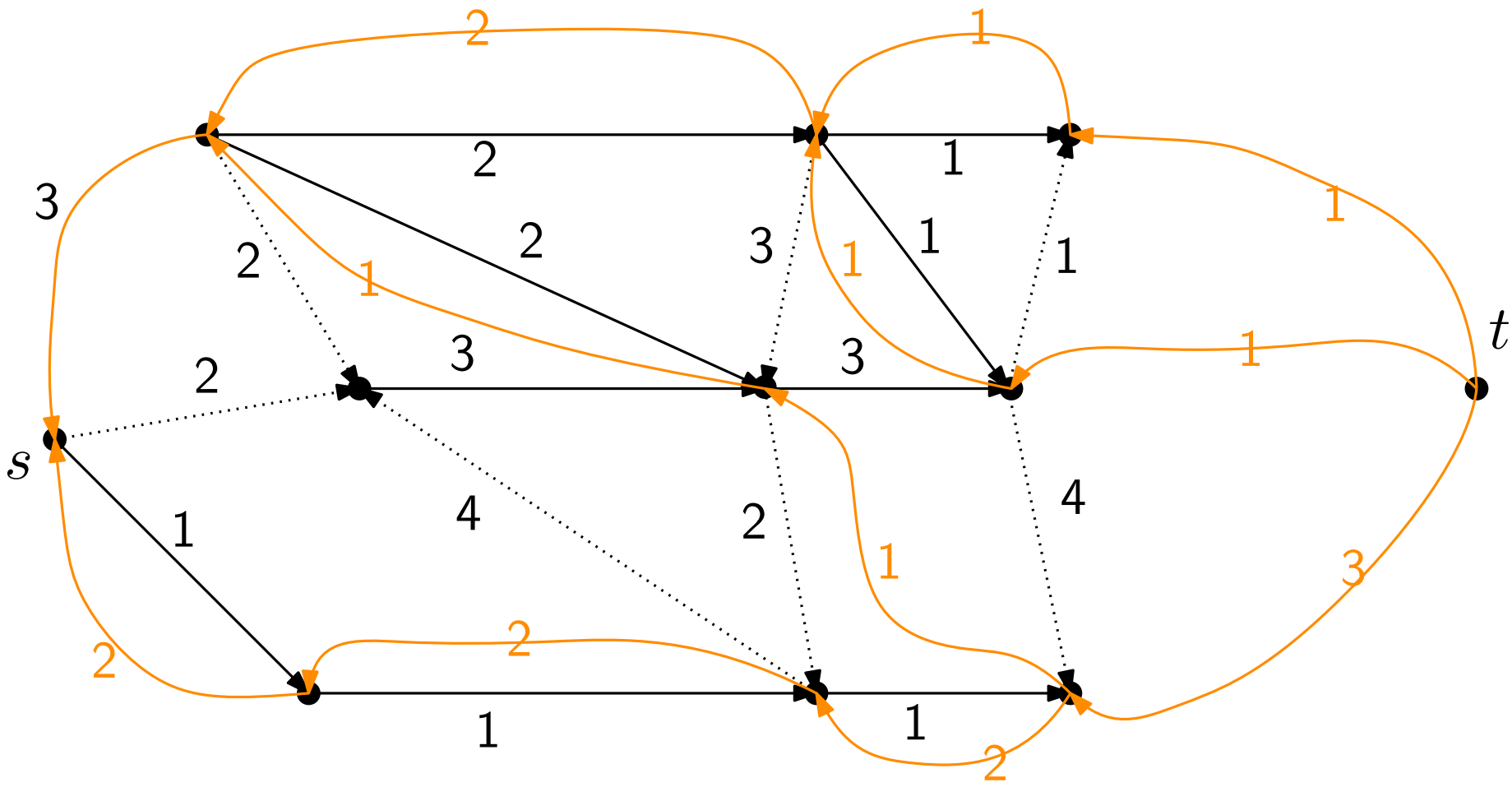


dist$(s, t) = 4$ in the original network.



dist$(s, t) = 4$ in the original network.

Back edges go backwards. Not part of any length-4-path.



dist$(s, t) = 4$ in the original network.

Back edges go backwards. Not part of any length-4-path.

dist$(s, t) > 4$ in the residual network.

Runtime Analysis

- dist(s, t) increases in every phase.

- dist(s, t) increases in every phase.

  At most $n - 2$ phases.

- dist(s, t) increases in every phase.

  At most $n - 2$ phases.
- Every phase performs a couple of depth-first searches.

- dist(s, t) increases in every phase.

  At most $n - 2$ phases.
- Every phase performs a couple of depth-first searches.

  How many?

- dist(s, t) increases in every phase.

  At most $n - 2$ phases.
- Every phase performs a couple of depth-first searches.

  How many?

  How much time does each take?

Claim. Each phase performs at most m depth-first searches.

Claim. Each phase performs at most m depth-first searches.

Proof.

Claim. Each phase performs at most m depth-first searches.

Proof.

- There are at most m edges in the network.

Claim. Each phase performs at most m depth-first searches.

Proof.

- There are at most m edges in the network.
- After every depth-first search we delete at least one edge.

Claim. Each phase performs at most m depth-first searches.

Proof.

- There are at most m edges in the network.
- After every depth-first search we delete at least one edge.

■

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Thus, the total running time of all (at most m) depth-first searches is:

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Thus, the total running time of all (at most m) depth-first searches is:

$O\left(\sum_{i=1}^{m}(n + d_i)\right)$

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Thus, the total running time of all (at most m) depth-first searches is:

$$O\left(\sum_{i=1}^{m}(n + d_i)\right) = O\left(nm + \sum_{i=1}^{m} d_i\right)$$

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Thus, the total running time of all (at most m) depth-first searches is:

$$\begin{aligned} O\left(\sum_{i=1}^{m}(n + d_i)\right) &= O\left(nm + \sum_{i=1}^{m} d_i\right) \\ &= O\left(nm + m\right) \end{aligned}$$

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Thus, the total running time of all (at most m) depth-first searches is:

$$\begin{aligned} O\left(\sum_{i=1}^{m}(n + d_i)\right) &= O\left(nm + \sum_{i=1}^{m} d_i\right) \\ &= O\left(nm + m\right) \\ &= O\left(nm\right) \end{aligned}$$

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Thus, the total running time of all (at most m) depth-first searches is:

$$\begin{aligned} O\left(\sum_{i=1}^{m}(n + d_i)\right) &= O\left(nm + \sum_{i=1}^{m} d_i\right) \\ &= O\left(nm + m\right) \\ &= O\left(nm\right) \end{aligned}$$

Thus, each phase takes $O(nm)$ steps.

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Thus, the total running time of all (at most m) depth-first searches is:

$$\begin{aligned} O\left(\sum_{i=1}^{m}(n + d_i)\right) &= O\left(nm + \sum_{i=1}^{m} d_i\right) \\ &= O\left(nm + m\right) \\ &= O\left(nm\right) \end{aligned}$$

Thus, each phase takes $O(nm)$ steps.

Dinic' algorithm performs at most $n - 2$ phases.

Observation. The i^{th} depth-first search takes time $O(n + d_i)$, where d_i is the number of edges deleted as dead ends.

Thus, the total running time of all (at most m) depth-first searches is:

$$\begin{aligned} O\left(\sum_{i=1}^{m}(n + d_i)\right) &= O\left(nm + \sum_{i=1}^{m} d_i\right) \\ &= O\left(nm + m\right) \\ &= O\left(nm\right) \end{aligned}$$

Thus, each phase takes $O(nm)$ steps.

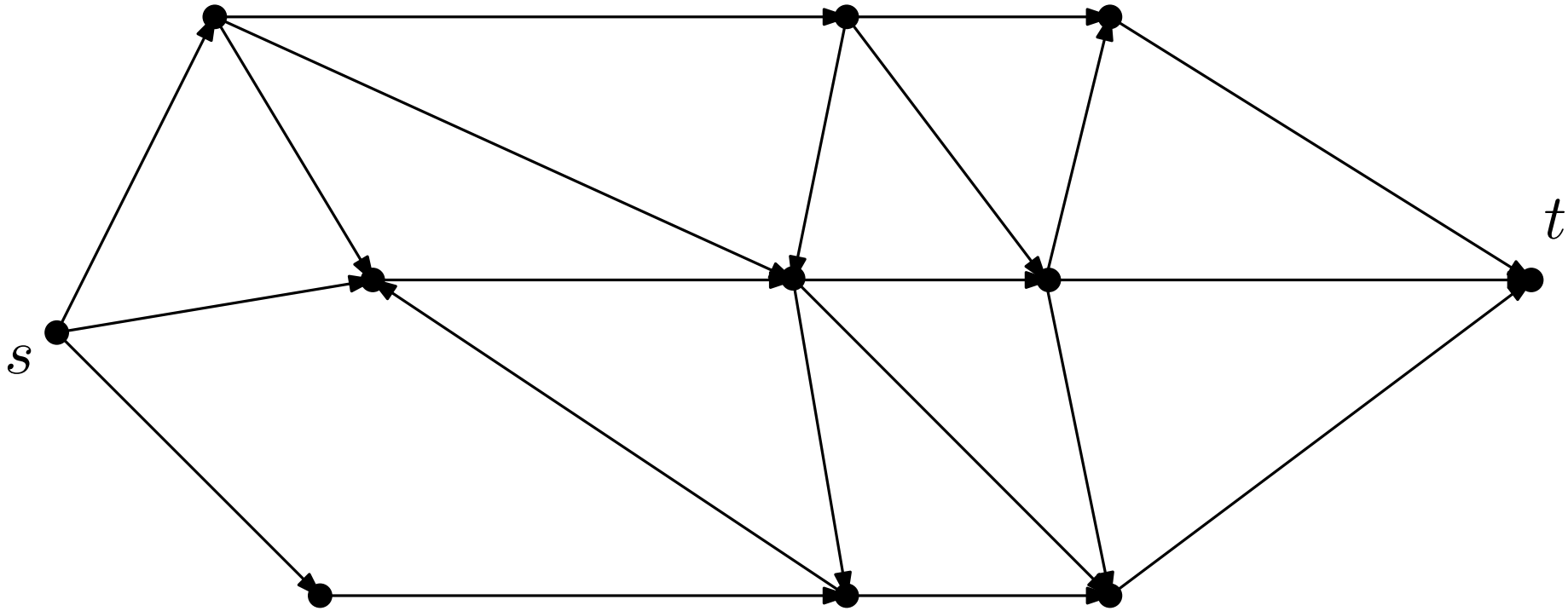
Dinic' algorithm performs at most $n - 2$ phases.

Dinic' algorithm performs $O(n^2 m)$ steps.

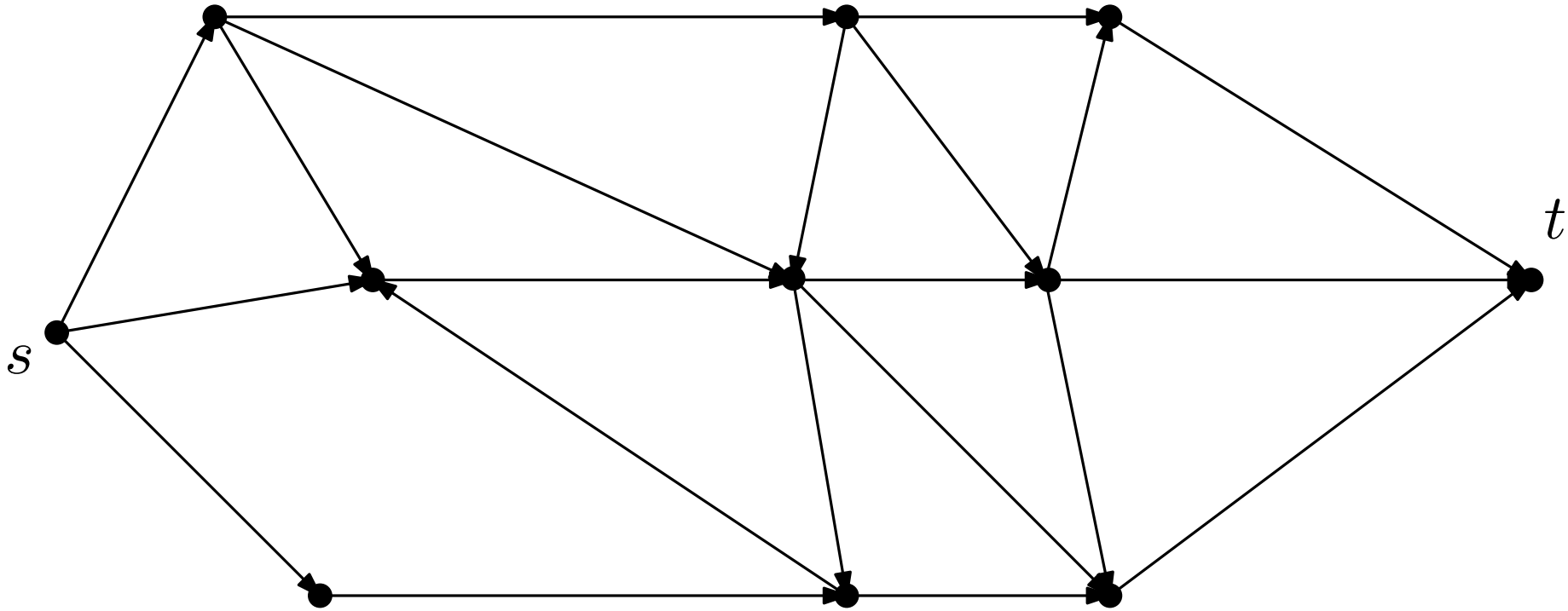
Theorem. Dinic' algorithm finds a maximum flow in $O(n^2m)$ steps.

Part II

Dinic' Algorithm in Unit Capacity Networks

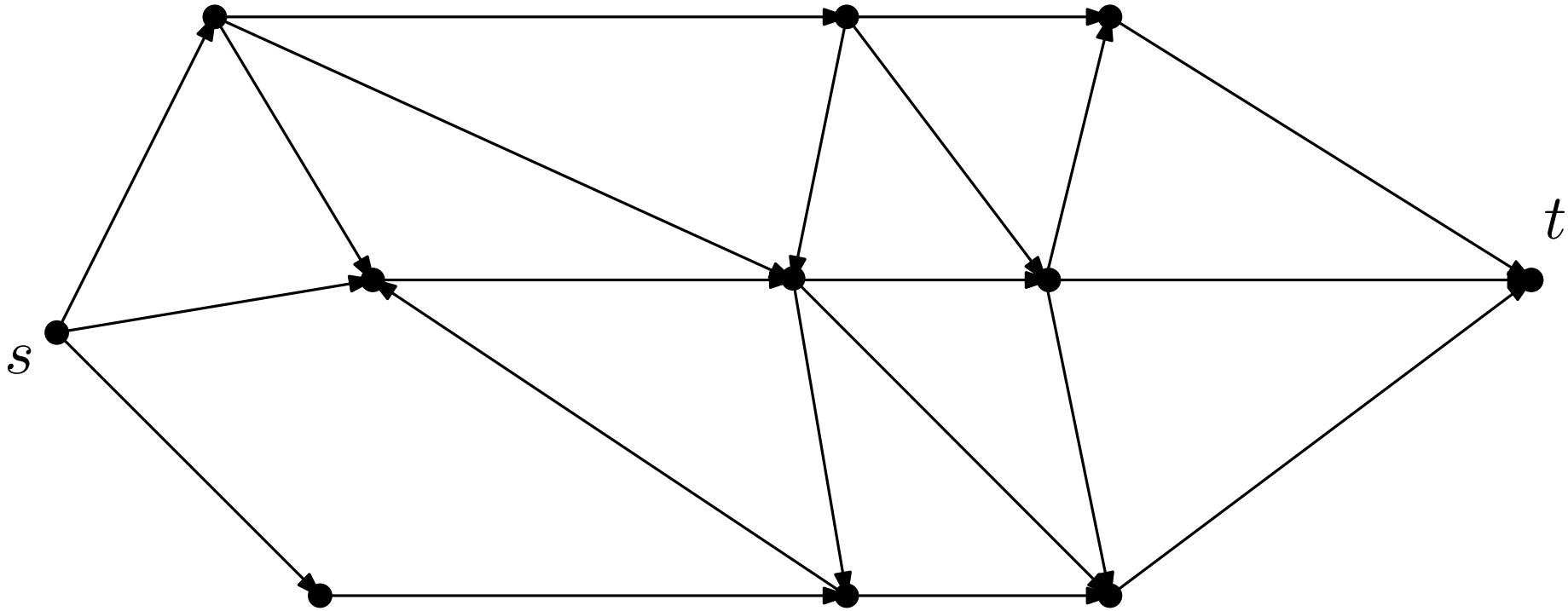


All capacities are 1



All capacities are 1

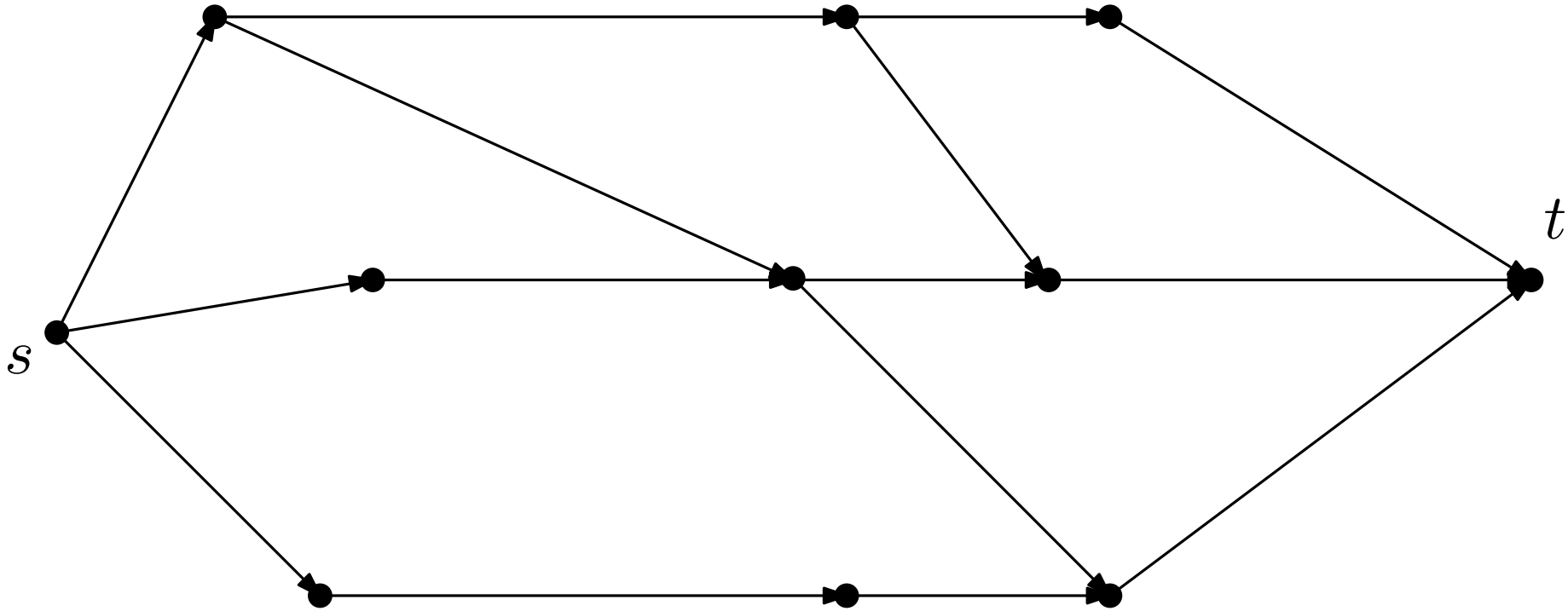
$k := \mathsf{dist}(s, t)$. Here, $k = 4$.



All capacities are 1

$k := \mathsf{dist}(s, t)$. Here, $k = 4$.

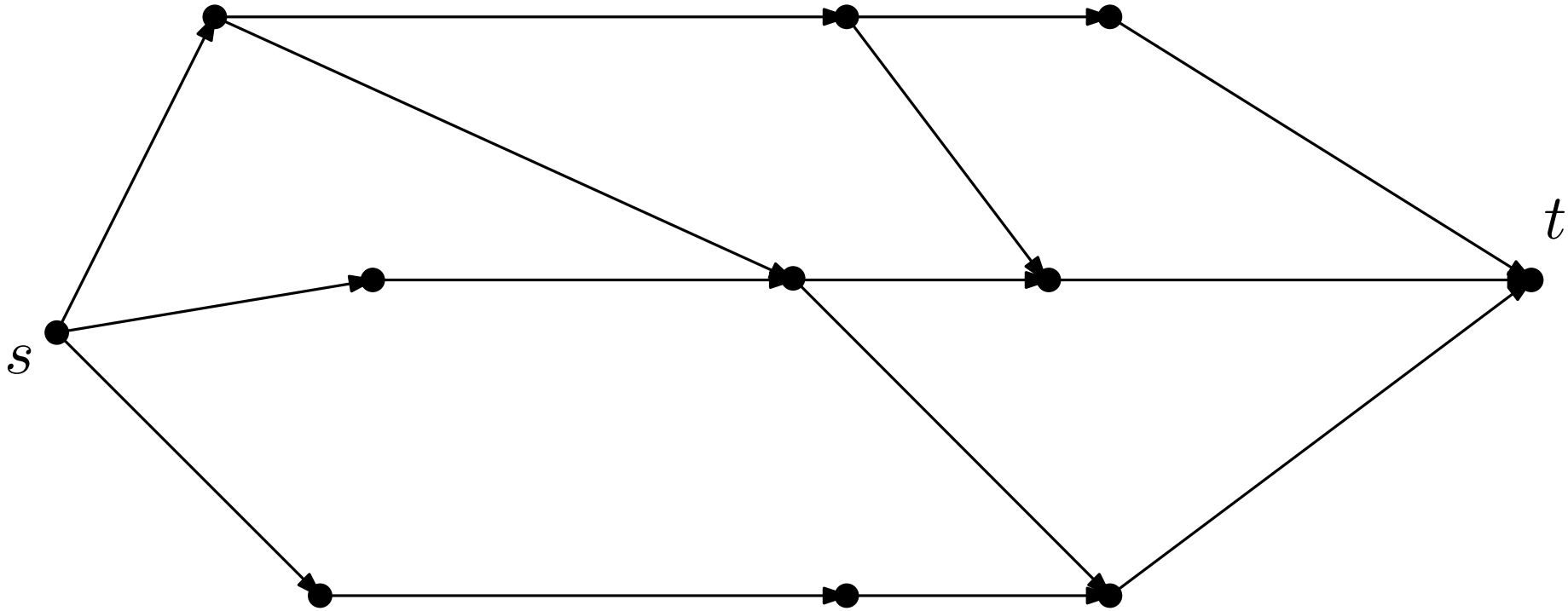
Ignore edges not on a shortest s-t-path.



All capacities are 1

$k := \mathsf{dist}(s, t)$. Here, $k = 4$.

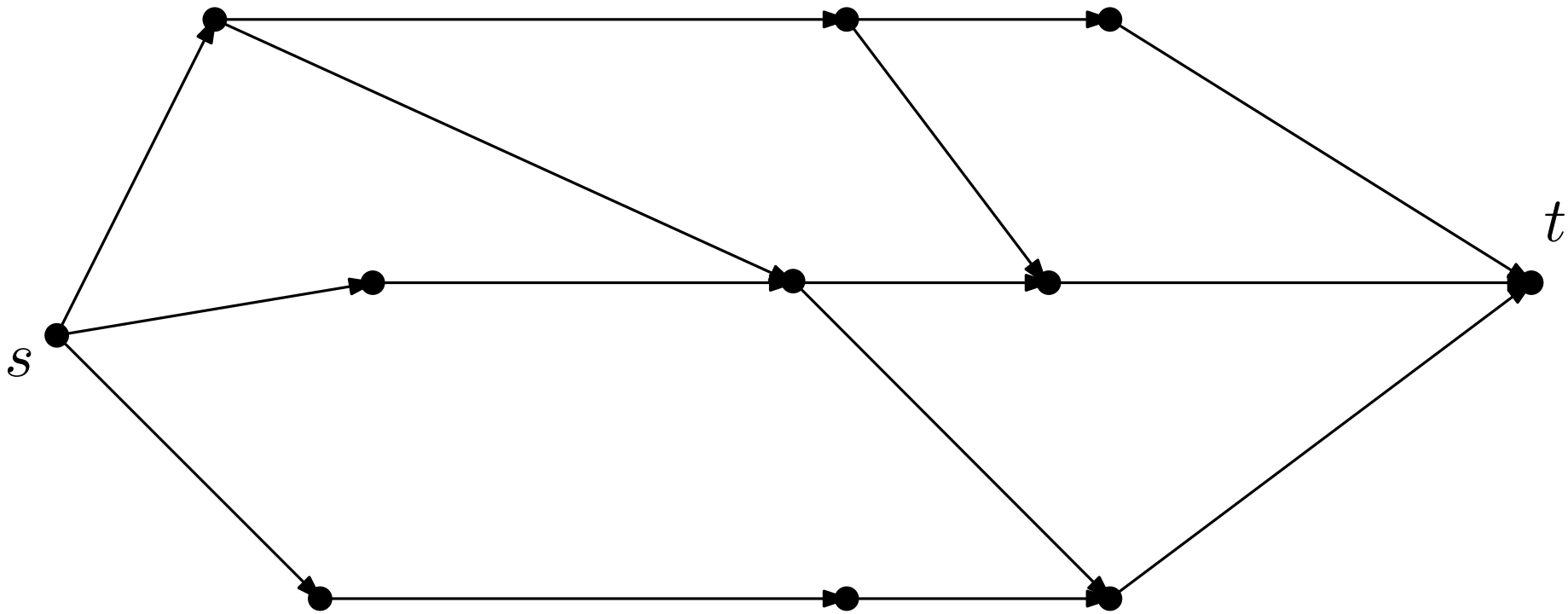
Ignore edges not on a shortest s-t-path.



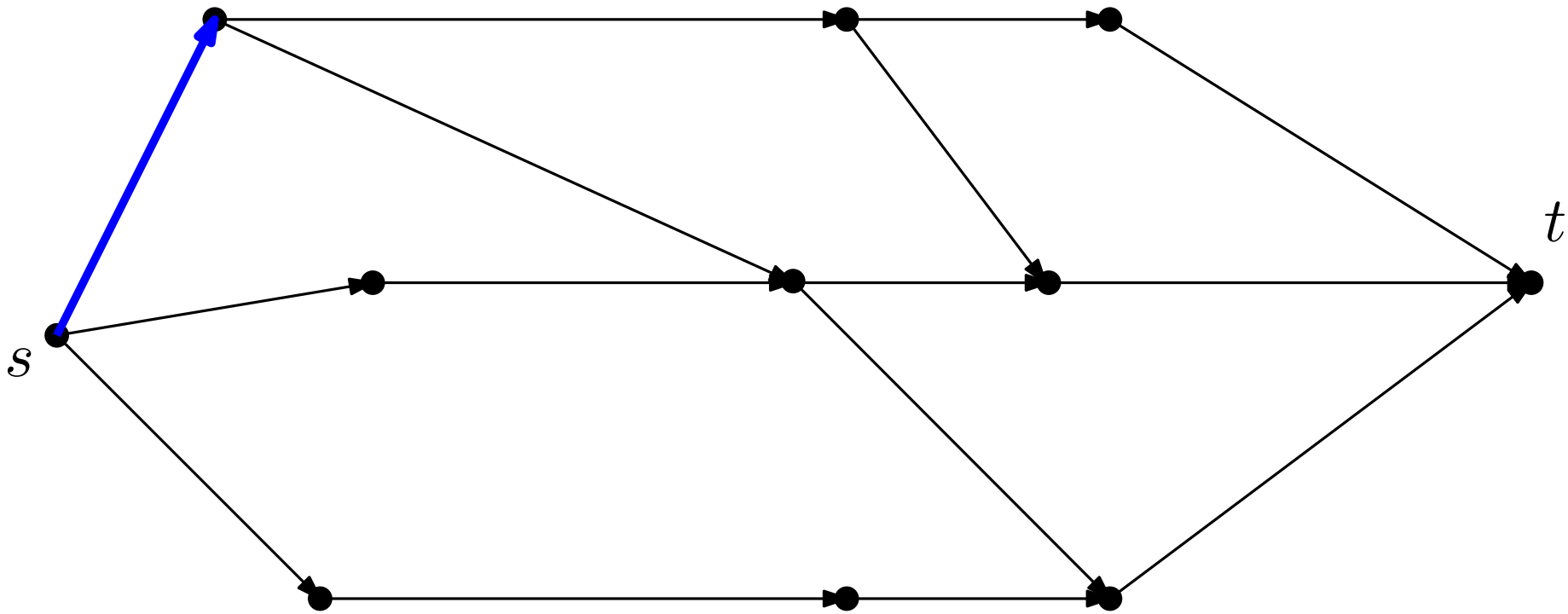
All capacities are 1

$k := \mathsf{dist}(s, t)$. Here, $k = 4$.

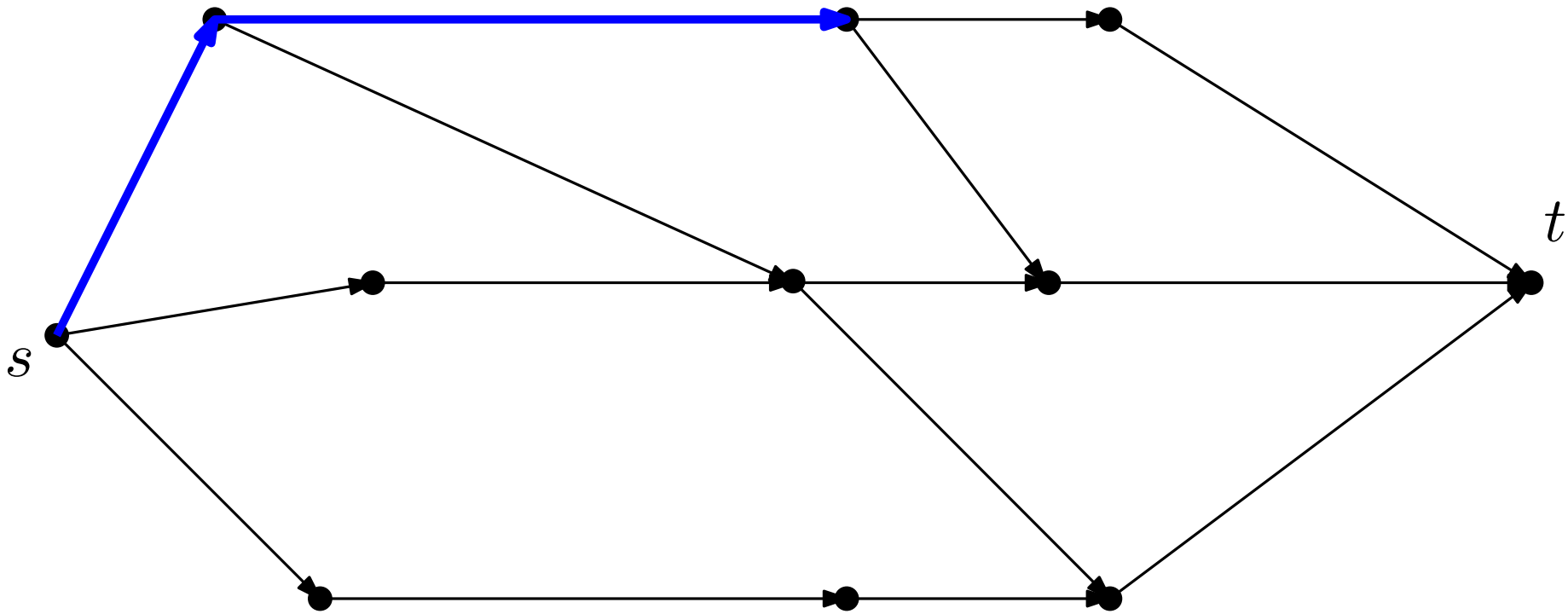
Ignore edges not on a shortest s-t-path.



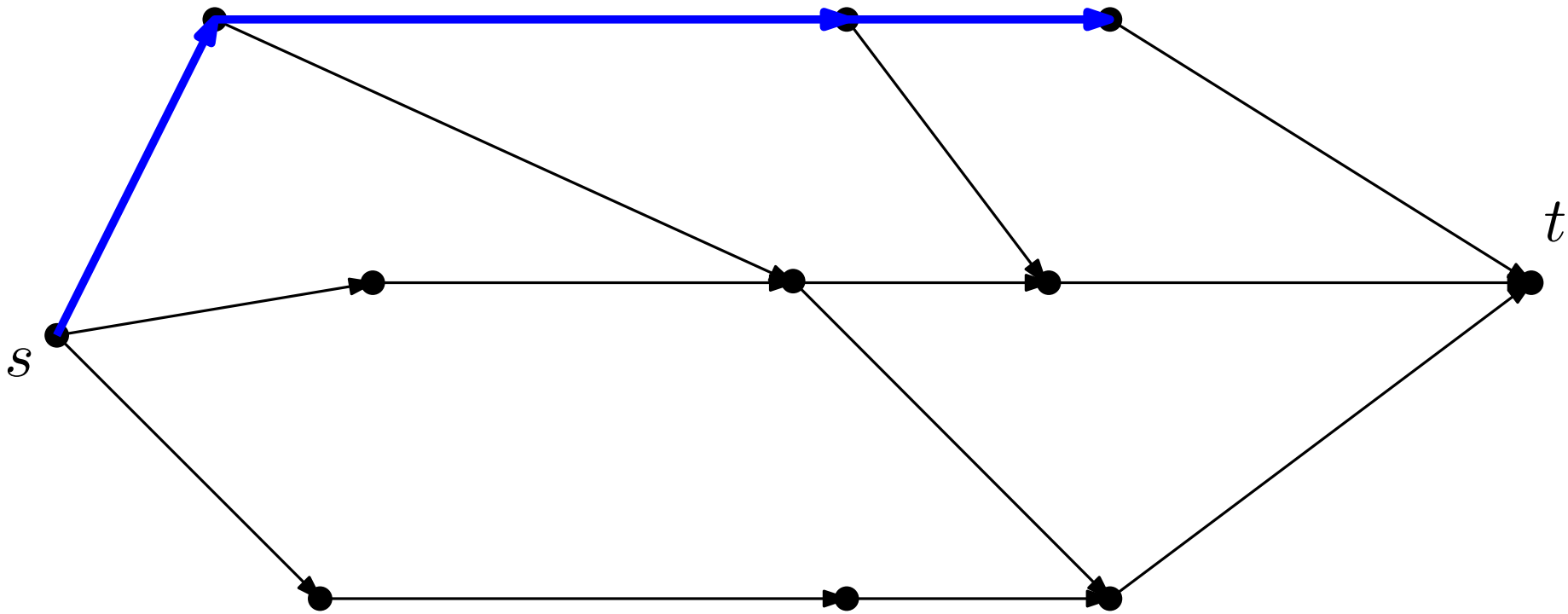
Find s-t-path by depth-first search.



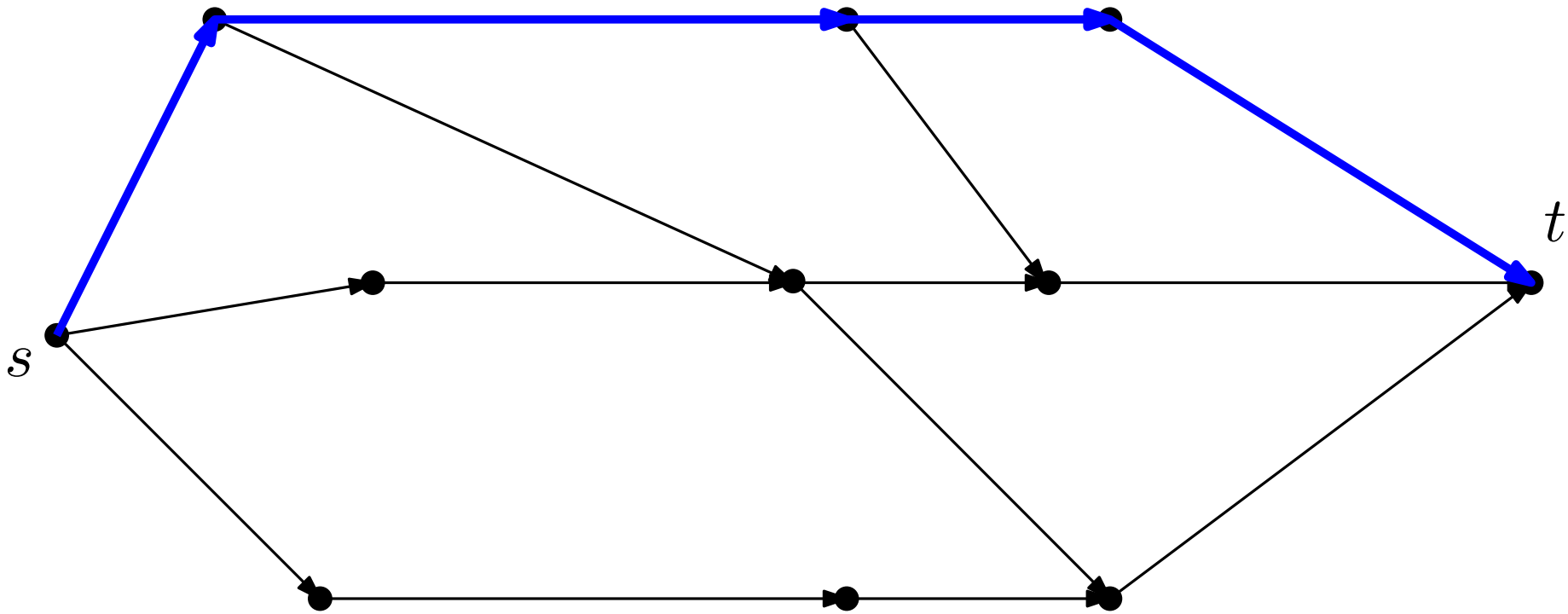
Find s-t-path by depth-first search.



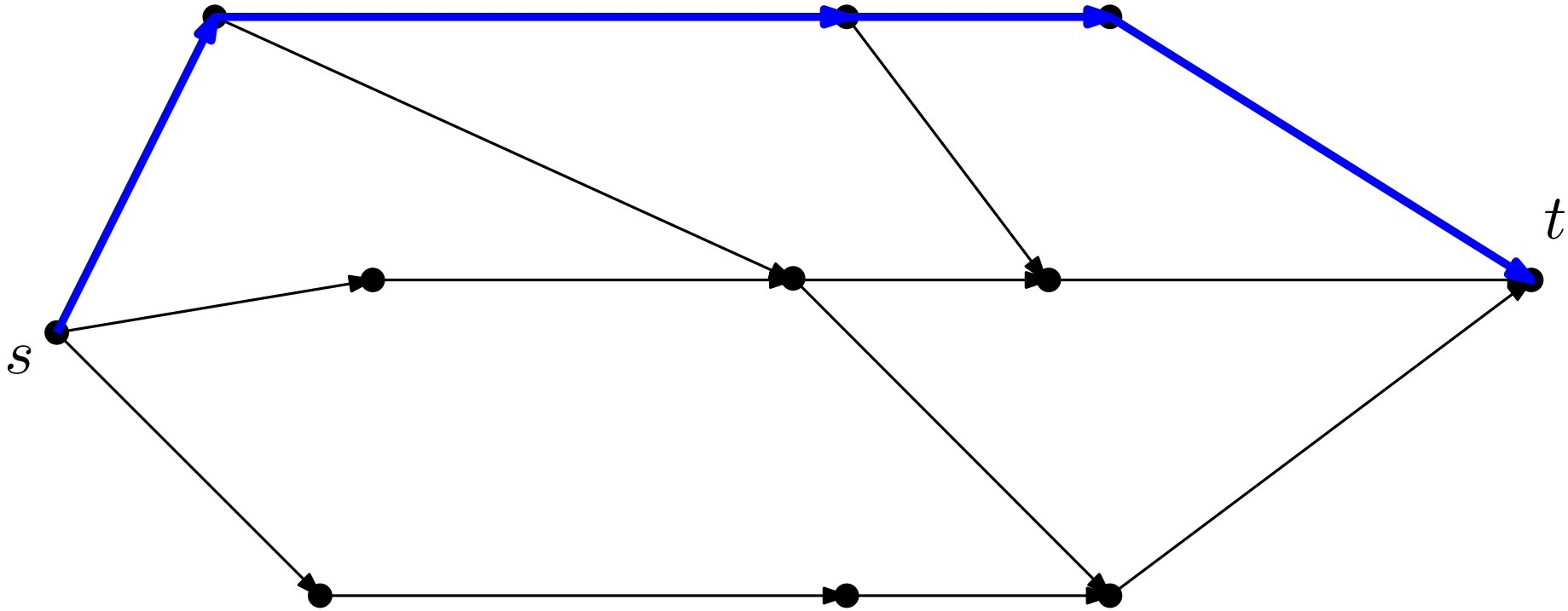
Find s-t-path by depth-first search.



Find s-t-path by depth-first search.

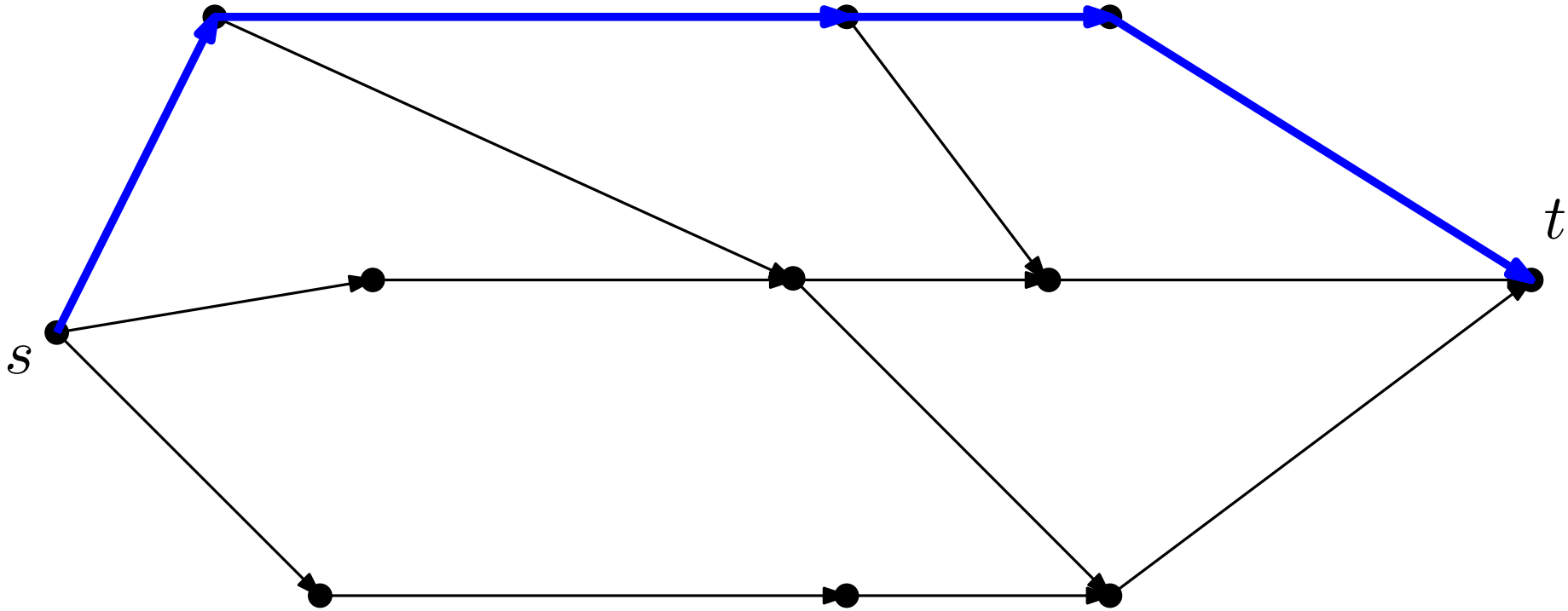


Find s-t-path by depth-first search.



Find s-t-path by depth-first search.

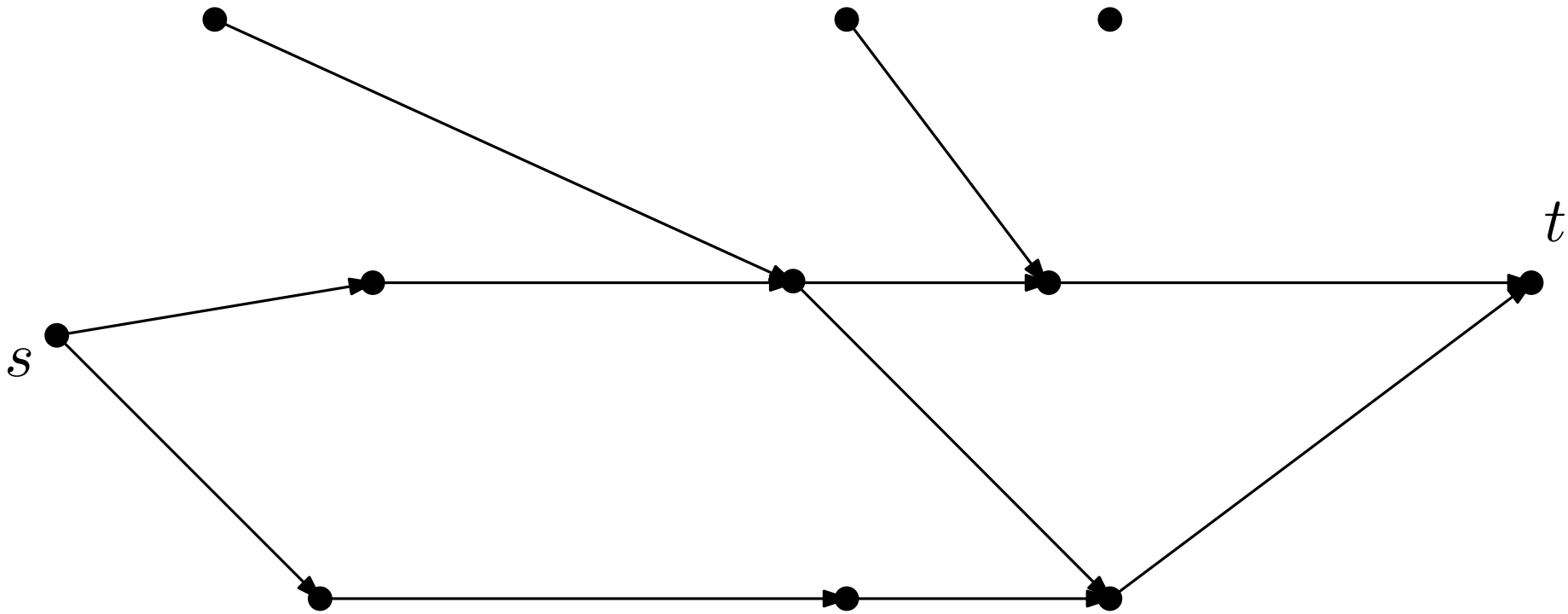
Route 1 unit of flow.



Find s-t-path by depth-first search.

Route 1 unit of flow.

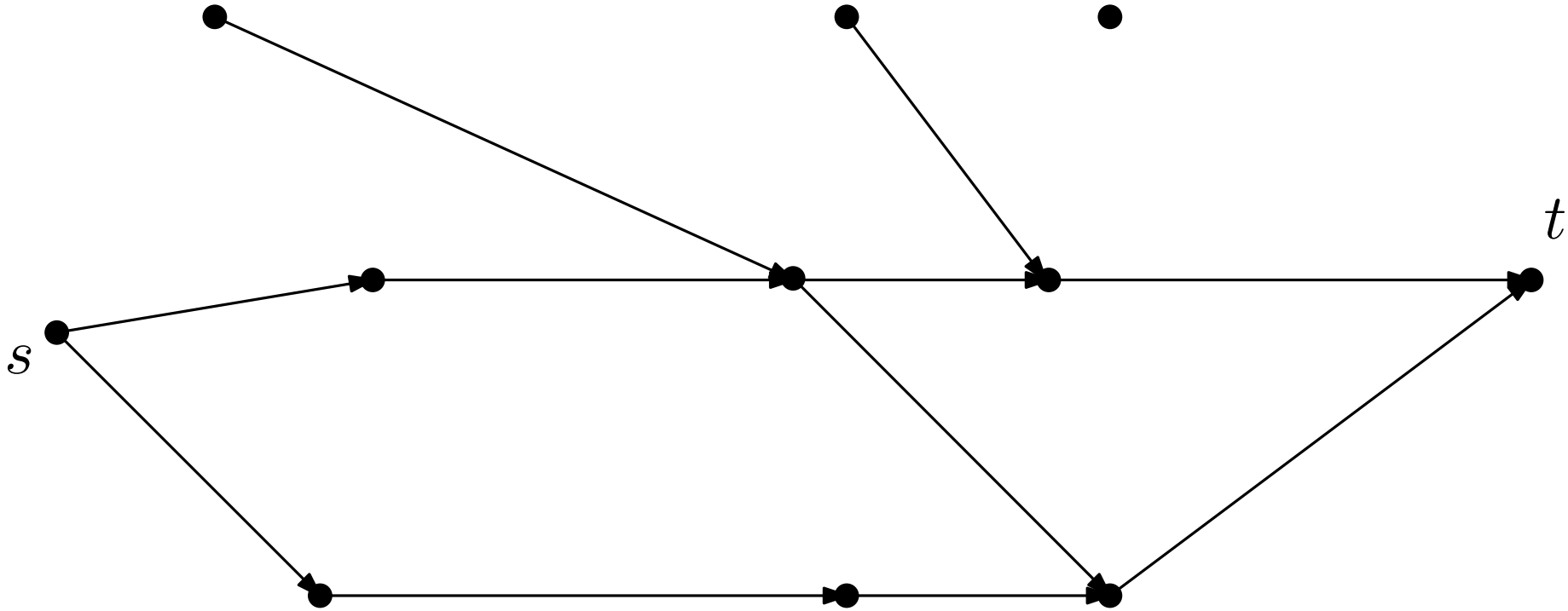
Update capacities.



Find s-t-path by depth-first search.

Route 1 unit of flow.

Update capacities.



Find s-t-path by depth-first search.

Route 1 unit of flow.

Update capacities. This deletes k edges!

Observation. Let $k = \text{dist}(s, t)$. Every capacity update removes k edges.

Observation. Let $k = \text{dist}(s, t)$. Every capacity update removes k edges.

Thus, each phase performs at most m/k depth first searches.

Observation. Let $k = \text{dist}(s, t)$. Every capacity update removes k edges.

Thus, each phase performs at most m/k depth first searches.

Every depth-first search takes time $O(k + d_i)$, where d_i is the number of edges deleted as dead ends.

Observation. Let $k = \text{dist}(s, t)$. Every capacity update removes k edges.

Thus, each phase performs at most m/k depth first searches.

Every depth-first search takes time $O(k + d_i)$, where d_i is the number of edges deleted as dead ends.

Number of steps performed in phase stage is:
$O\left(\sum_{i=1}^{m/k}(k + d_i)\right)$

Observation. Let $k = \text{dist}(s, t)$. Every capacity update removes k edges.

Thus, each phase performs at most m/k depth first searches.

Every depth-first search takes time $O(k + d_i)$, where d_i is the number of edges deleted as dead ends.

Number of steps performed in phase stage is:
$O\left(\sum_{i=1}^{m/k}(k + d_i)\right) = O\left(m + \sum_{i=1}^{m/k} d_i\right)$

Observation. Let $k = \text{dist}(s, t)$. Every capacity update removes k edges.

Thus, each phase performs at most m/k depth first searches.

Every depth-first search takes time $O(k + d_i)$, where d_i is the number of edges deleted as dead ends.

Number of steps performed in phase stage is:
$O\left(\sum_{i=1}^{m/k}(k + d_i)\right) = O\left(m + \sum_{i=1}^{m/k} d_i\right) = O(m)$.

Observation. Let $k = \text{dist}(s, t)$. Every capacity update removes k edges.

Thus, each phase performs at most m/k depth first searches.

Every depth-first search takes time $O(k + d_i)$, where d_i is the number of edges deleted as dead ends.

Number of steps performed in phase stage is:
$O\left(\sum_{i=1}^{m/k}(k + d_i)\right) = O\left(m + \sum_{i=1}^{m/k} d_i\right) = O(m)$.

There are at most $n - 2$ phases.

Theorem. On a network in which all capacities are 1, Dinic' algorithm finds a maximum flow in $O(nm)$ steps.

Theorem. On a network in which all capacities are 1, Dinic' algorithm finds a maximum flow in $O(nm)$ steps.

even better:

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

Theorem. On a network in which all capacities are 1, Dinic' algorithm finds a maximum flow in $O(nm)$ steps.

even better:

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

Lemma 2. On a network with unit capacities, Dinic' algorithm performs at most $O\left(n^{2/3}\right)$ phases, and thus $O(n^{2/3}m)$ steps.

Part III

Number of Phases in Unit Capacity Networks

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $2\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $2\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

Proof. Let $t \in \mathbf{N}$, to be determined later.

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $2\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

Proof. Let $t \in \mathbf{N}$, to be determined later.

Consider G_t, the residual network after t phases.

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $2\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

Proof. Let $t \in \mathbf{N}$, to be determined later.

Consider G_t, the residual network after t phases.

$k := \mathsf{dist}(s, t) \geq t$ in G_t.

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $2\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

Proof. Let $t \in \mathbf{N}$, to be determined later.

Consider G_t, the residual network after t phases.

$k := \mathsf{dist}(s, t) \geq t$ in G_t.

$V_i := \{v \mid \mathsf{dist}(s, v) = i \text{ in } G_t\}$.

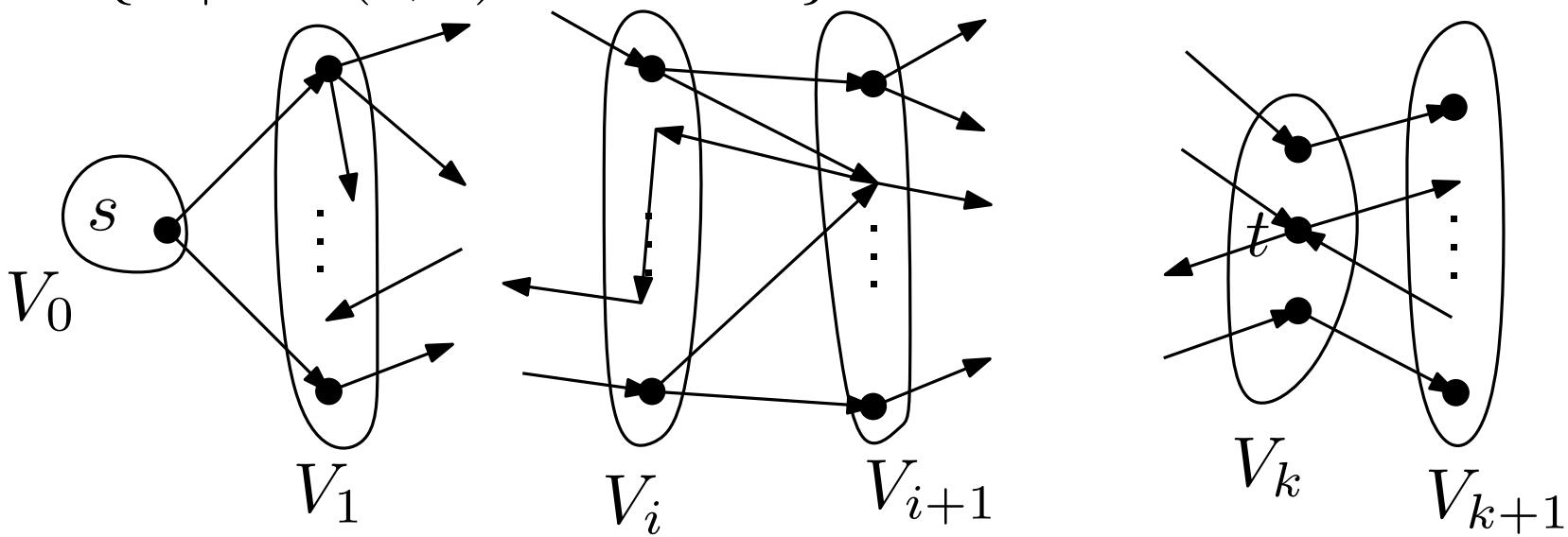
Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $2\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

Proof. Let $t \in \mathbf{N}$, to be determined later.

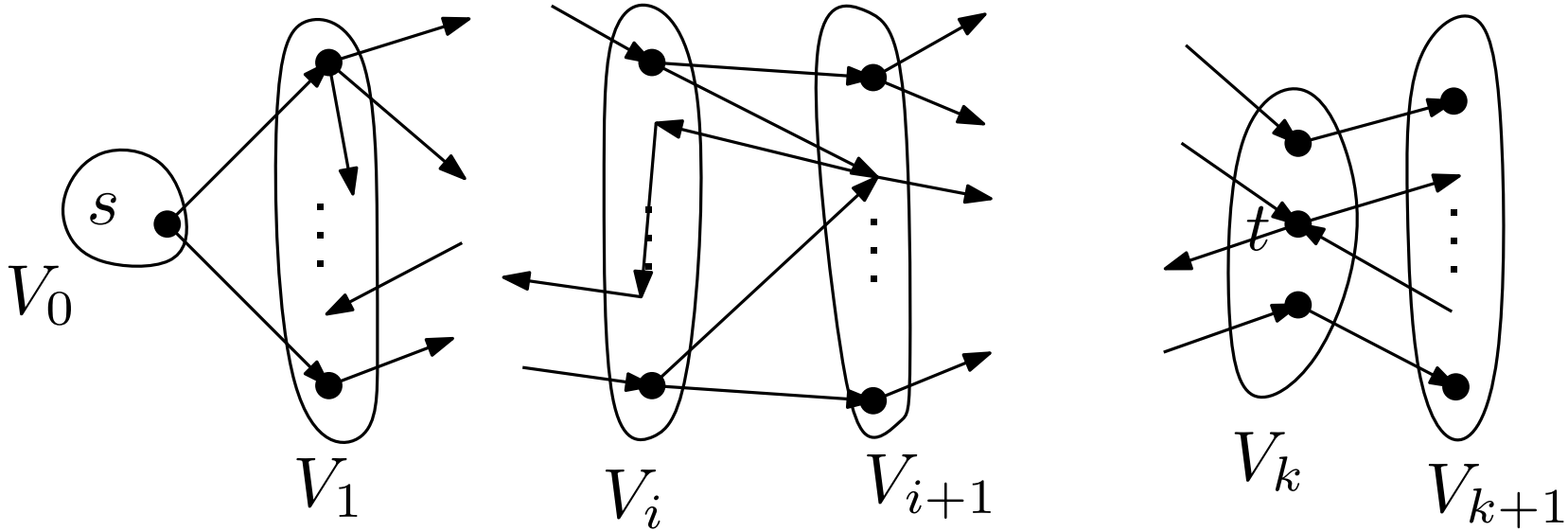
Consider G_t, the residual network after t phases.

$k := \mathsf{dist}(s, t) \geq t$ in G_t.

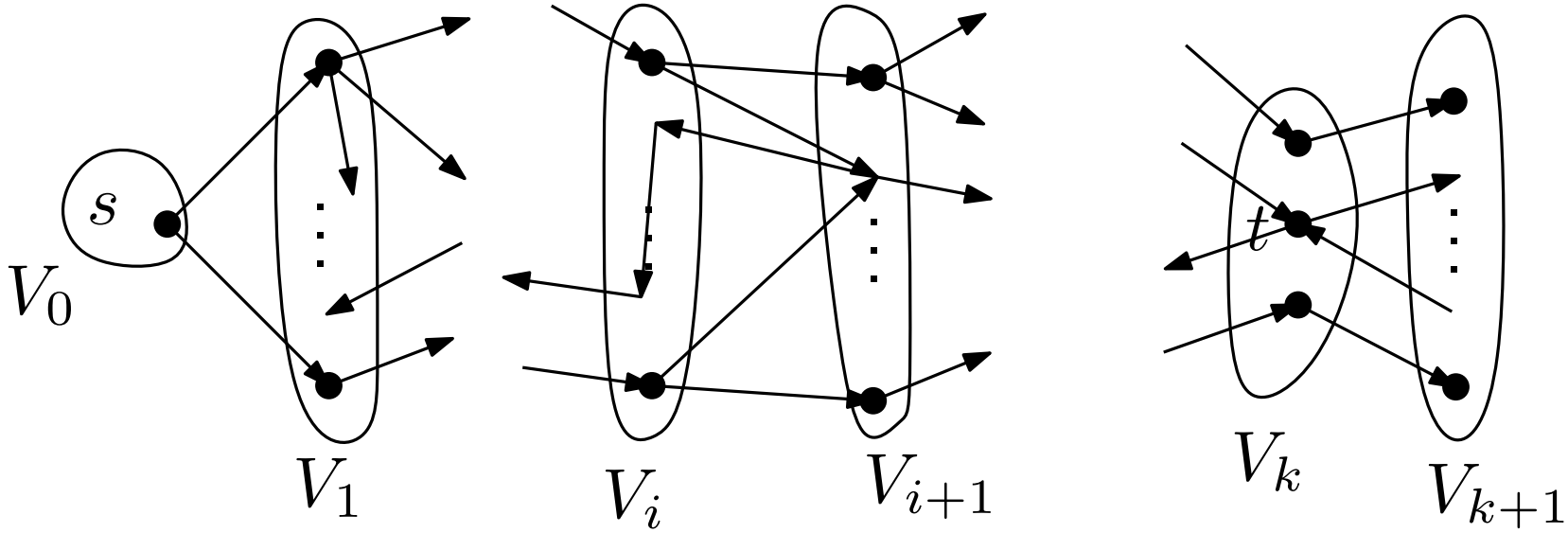
$V_i := \{v \mid \mathsf{dist}(s, v) = i \text{ in } G_t\}$.



$V_i := \{v \mid \mathsf{dist}(s, v) = i \text{ in } G_t\}$.

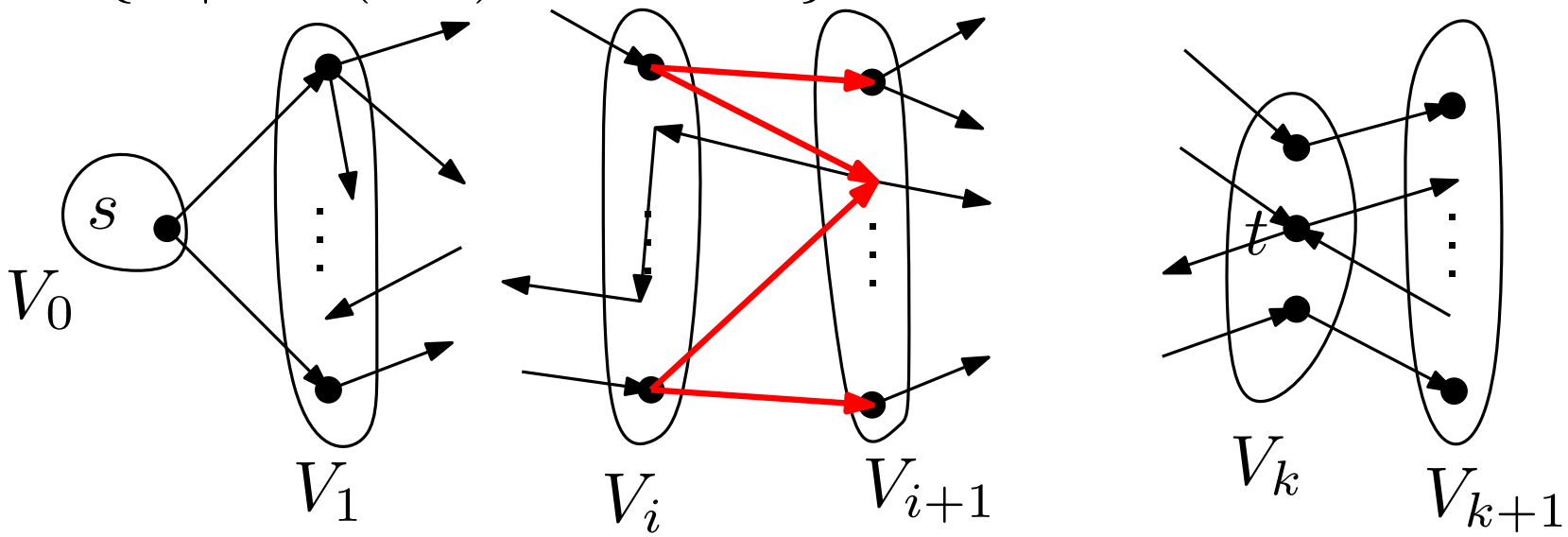


$V_i := \{v \mid \mathsf{dist}(s, v) = i \text{ in } G_t\}.$



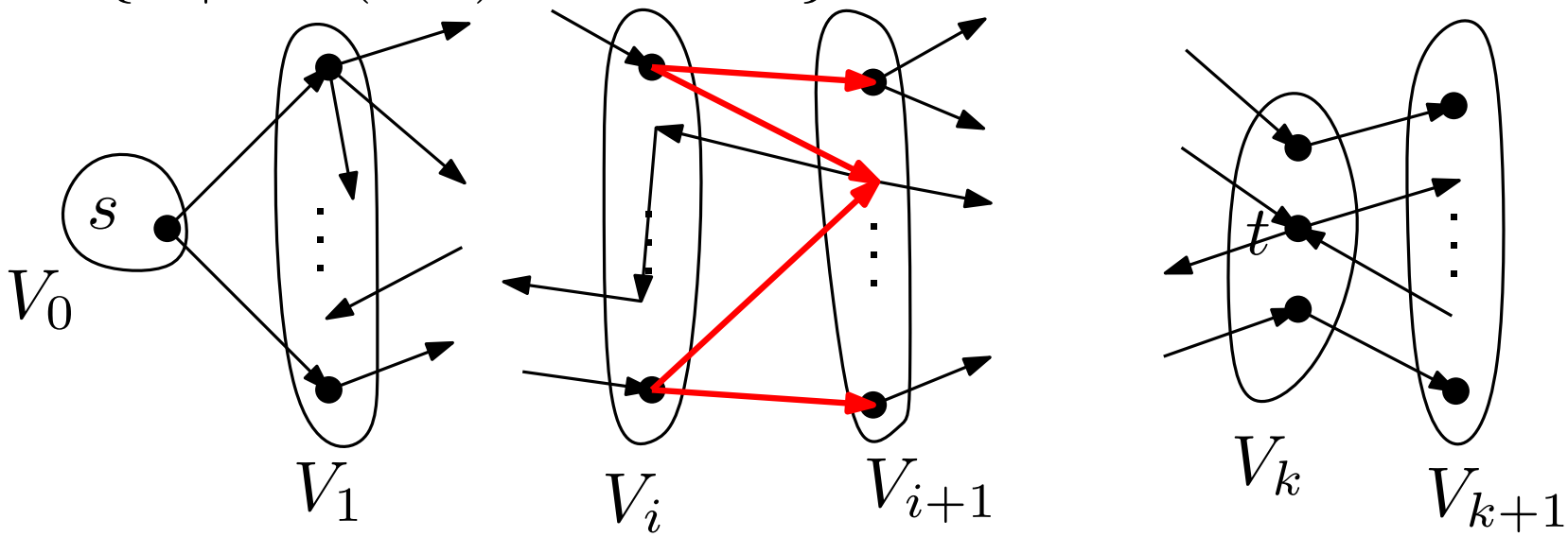
$E_i := \{(u, v) \in E \mid u \in V_i, v \in V_{i+1}\}$

$V_i := \{v \mid \mathsf{dist}(s, v) = i \text{ in } G_t\}.$



$E_i := \{(u, v) \in E \mid u \in V_i, v \in V_{i+1}\}$

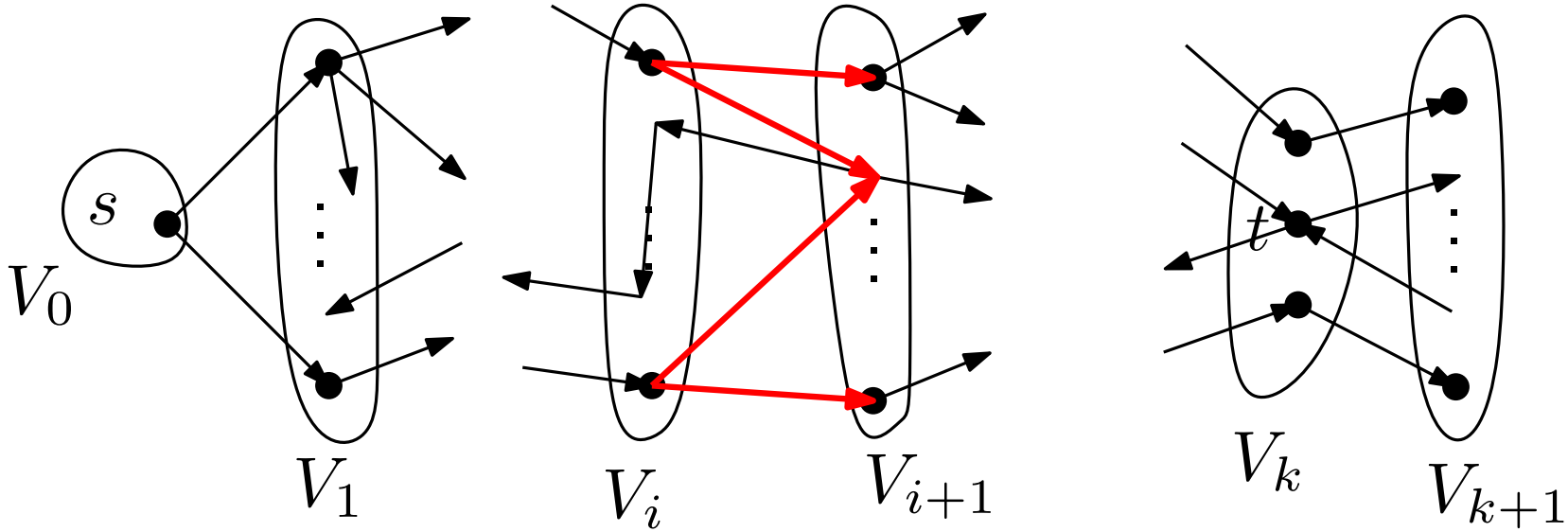
$V_i := \{v \mid \mathsf{dist}(s, v) = i \text{ in } G_t\}.$



$E_i := \{(u, v) \in E \mid u \in V_i, v \in V_{i+1}\}$

$|E_0| + |E_1| + \ldots + |E_{k-1}| \leq m.$

$V_i := \{v \mid \mathsf{dist}(s, v) = i \text{ in } G_t\}$.

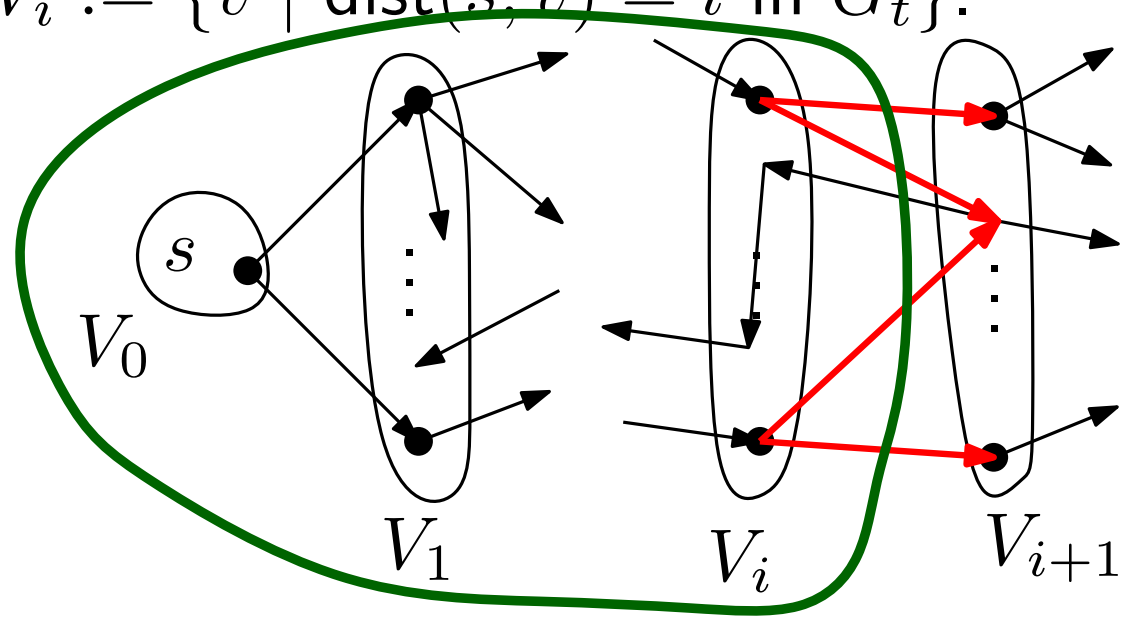


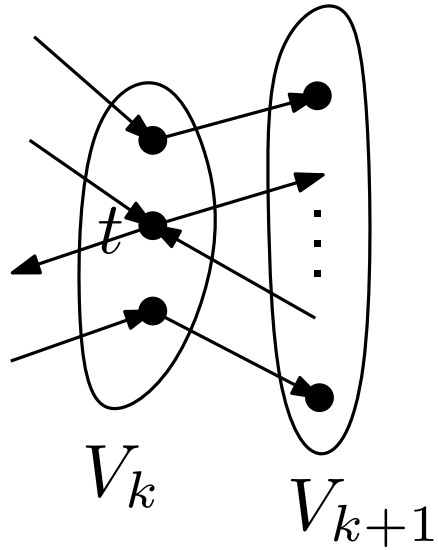
$E_i := \{(u, v) \in E \mid u \in V_i, v \in V_{i+1}\}$

$|E_0| + |E_1| + \ldots + |E_{k-1}| \leq m$.

$|E_i| \leq \frac{m}{k} \leq \frac{m}{t}$ for some i.

$V_i := \{v \mid \text{dist}(s, v) = i \text{ in } G_t\}$.





$E_i := \{(u, v) \in E \mid u \in V_i, v \in V_{i+1}\}$

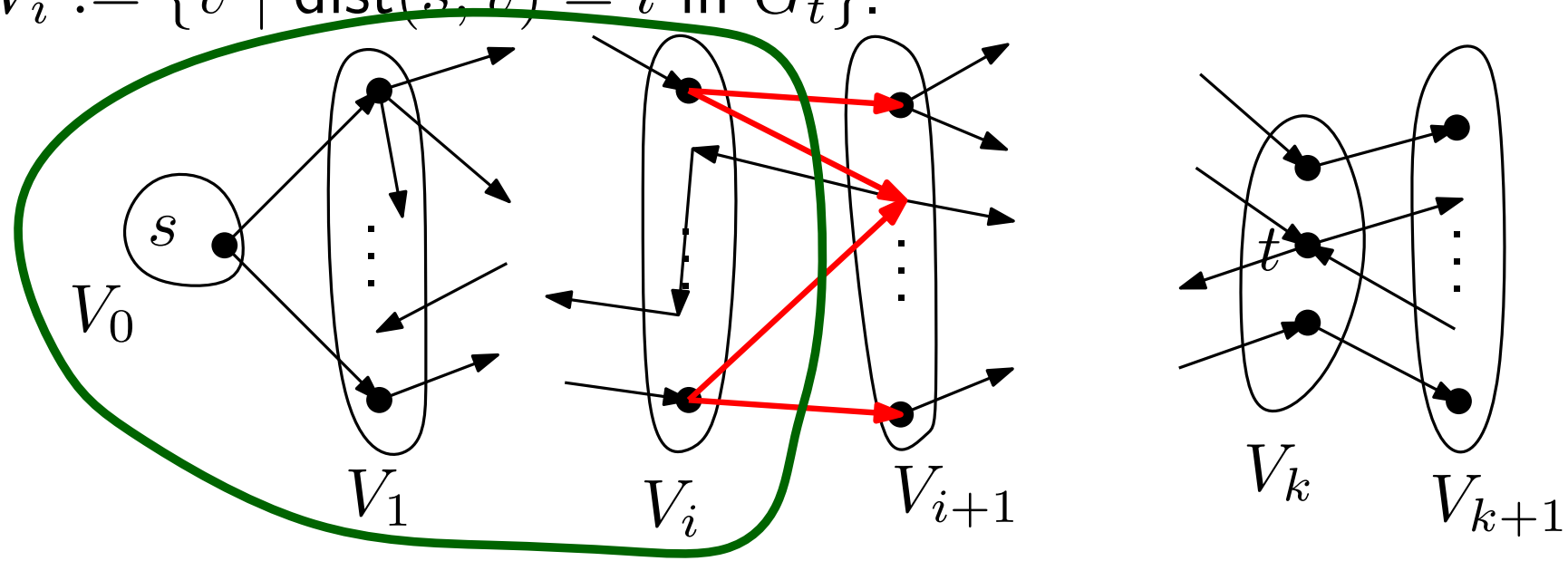
$|E_0| + |E_1| + \ldots + |E_{k-1}| \leq m.$

$|E_i| \leq \frac{m}{k} \leq \frac{m}{t}$ for some i.

$S := V_0 \cup V_1 \cup \ldots \cup V_i.$

is an s-t-cut!

$V_i := \{v \mid \text{dist}(s, v) = i \text{ in } G_t\}$.



$E_i := \{(u, v) \in E \mid u \in V_i, v \in V_{i+1}\}$

$|E_0| + |E_1| + \ldots + |E_{k-1}| \leq m$.

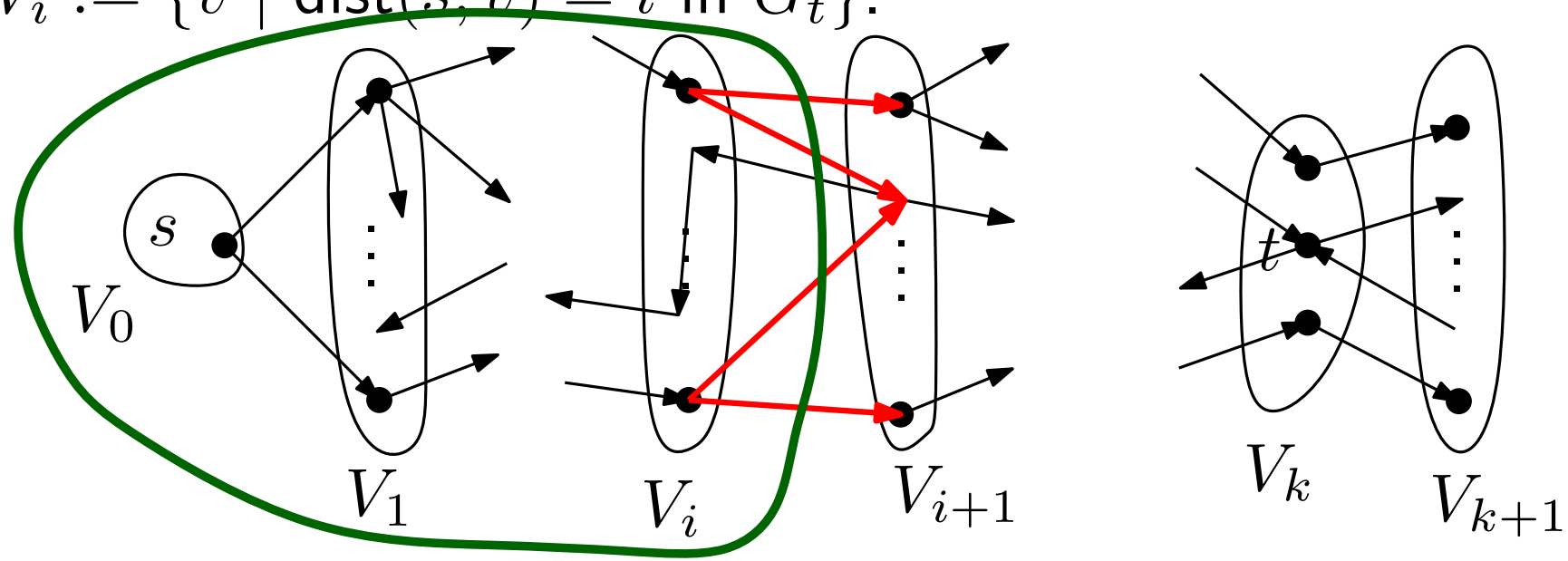
$|E_i| \leq \frac{m}{k} \leq \frac{m}{t}$ for some i.

$\text{cap}(S, V \setminus S) = |E_i| \leq \frac{m}{t}$.

$S := V_0 \cup V_1 \cup \ldots \cup V_i$.

is an s-t-cut!

$V_i := \{v \mid \text{dist}(s, v) = i \text{ in } G_t\}$.



$E_i := \{(u, v) \in E \mid u \in V_i, v \in V_{i+1}\}$

$|E_0| + |E_1| + \ldots + |E_{k-1}| \leq m$.

$|E_i| \leq \frac{m}{k} \leq \frac{m}{t}$ for some i.

$S := V_0 \cup V_1 \cup \ldots \cup V_i$.

is an s-t-cut!

$\text{cap}(S, V \setminus S) = |E_i| \leq \frac{m}{t}$.

$\text{maxflow}(G_t) \leq \frac{m}{t}$.

$G_t :=$ the residual network after t phases.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{m}{t}$.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{m}{t}$.

Every phase increases the flow by at least 1.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{m}{t}$.

Every phase increases the flow by at least 1.

At most $t + \frac{m}{t}$ phases.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{m}{t}$.

Every phase increases the flow by at least 1.

At most $t + \frac{m}{t}$ phases.

At most $2\sqrt{m}$ phases (setting $t = \sqrt{m}$).

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{m}{t}$.

Every phase increases the flow by at least 1.

At most $t + \frac{m}{t}$ phases.

At most $2\sqrt{m}$ phases (setting $t = \sqrt{m}$).

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $2\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{m}{t}$.

Every phase increases the flow by at least 1.

At most $t + \frac{m}{t}$ phases.

At most $2\sqrt{m}$ phases (setting $t = \sqrt{m}$).

Lemma 1. On a network with unit capacities, Dinic' algorithm performs at most $2\sqrt{m}$ phases, and thus $O(m^{1.5})$ steps.

■

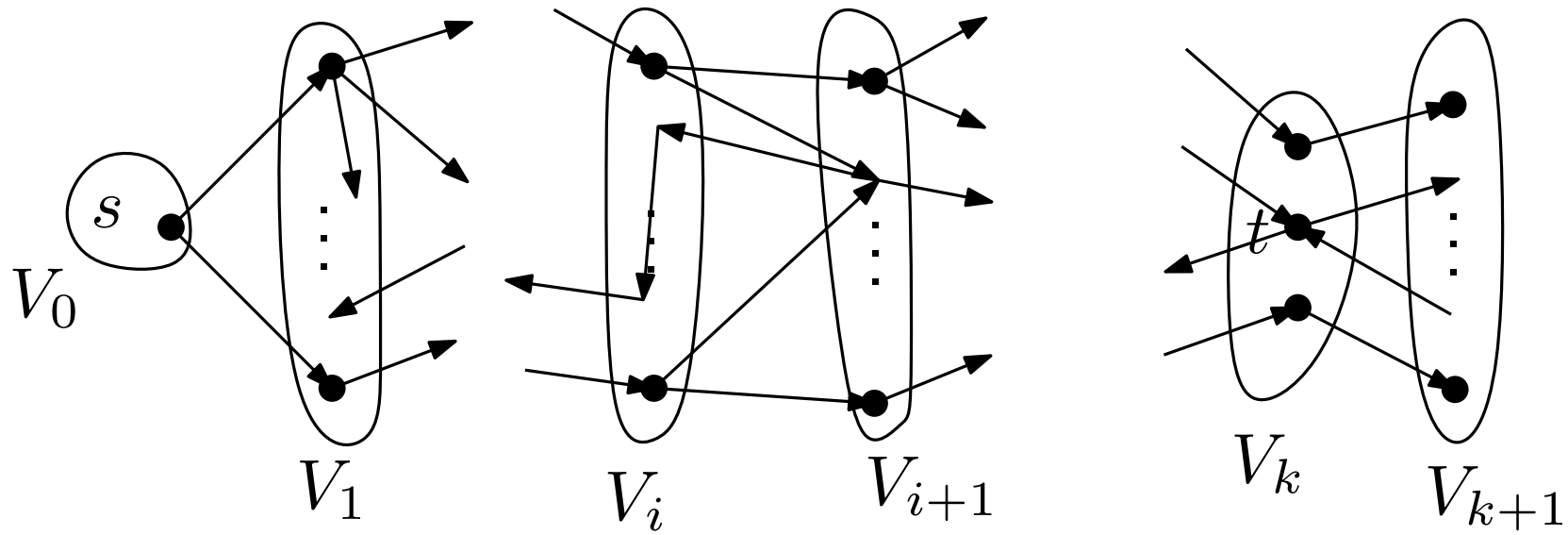
Lemma 2. On a network with unit capacities, Dinic' algorithm performs at most $O\left(n^{2/3}\right)$ phases, and thus $O(n^{2/3}m)$ steps.

Lemma 2. On a network with unit capacities, Dinic' algorithm performs at most $O\left(n^{2/3}\right)$ phases, and thus $O(n^{2/3}m)$ steps.

Proof.

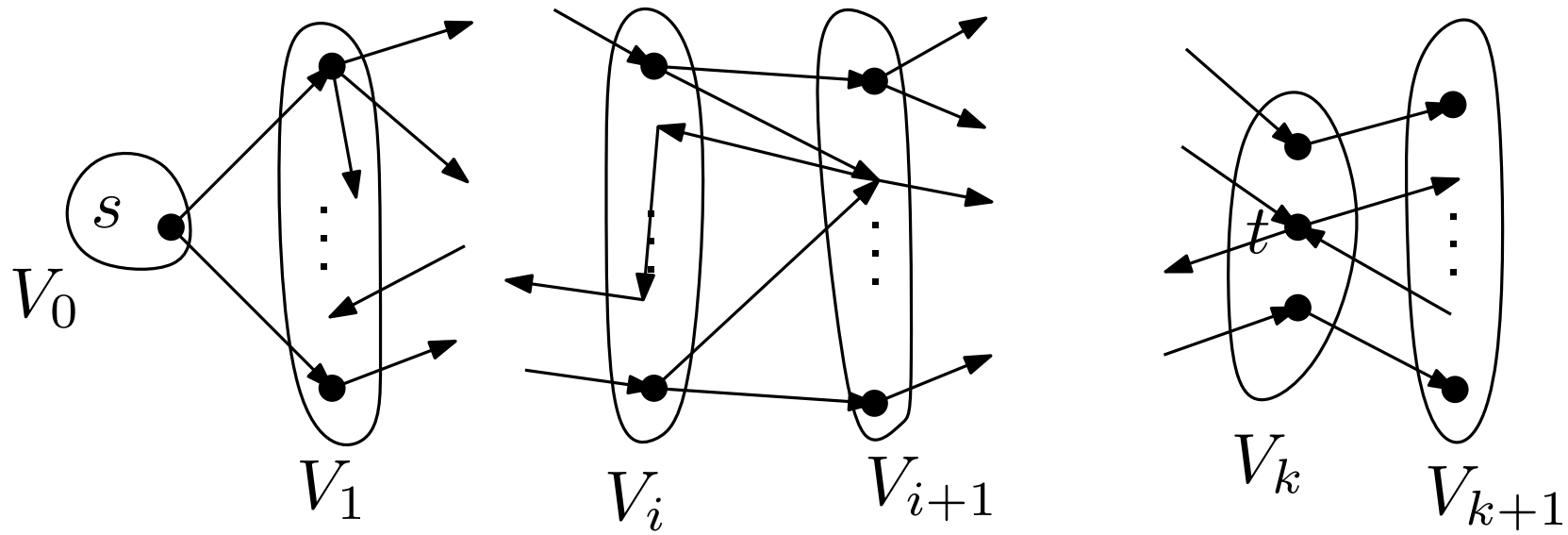
Lemma 2. On a network with unit capacities, Dinic' algorithm performs at most $O\left(n^{2/3}\right)$ phases, and thus $O(n^{2/3}m)$ steps.

Proof. $V_i := \{v \mid \text{dist}(s, v) = i \text{ in } G_t\}$.



Lemma 2. On a network with unit capacities, Dinic' algorithm performs at most $O\left(n^{2/3}\right)$ phases, and thus $O(n^{2/3}m)$ steps.

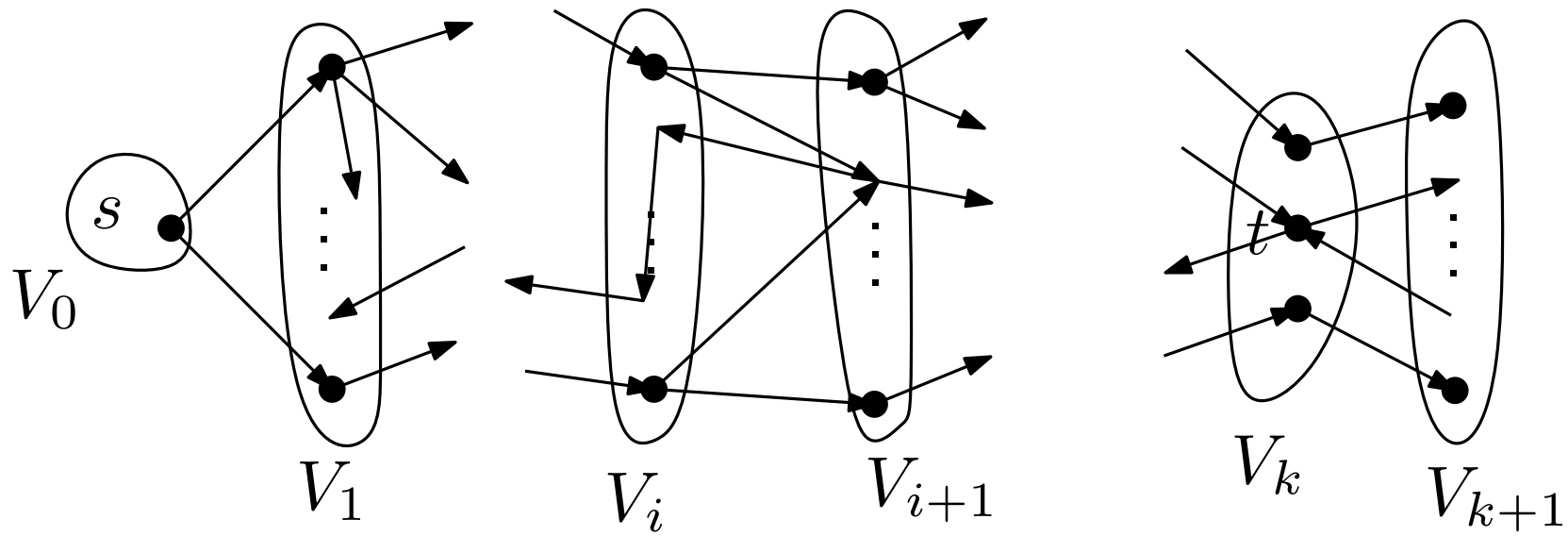
Proof. $V_i := \{v \mid \text{dist}(s, v) = i \text{ in } G_t\}$.



V_i is *big* if $|V_i| > 2n/k$.

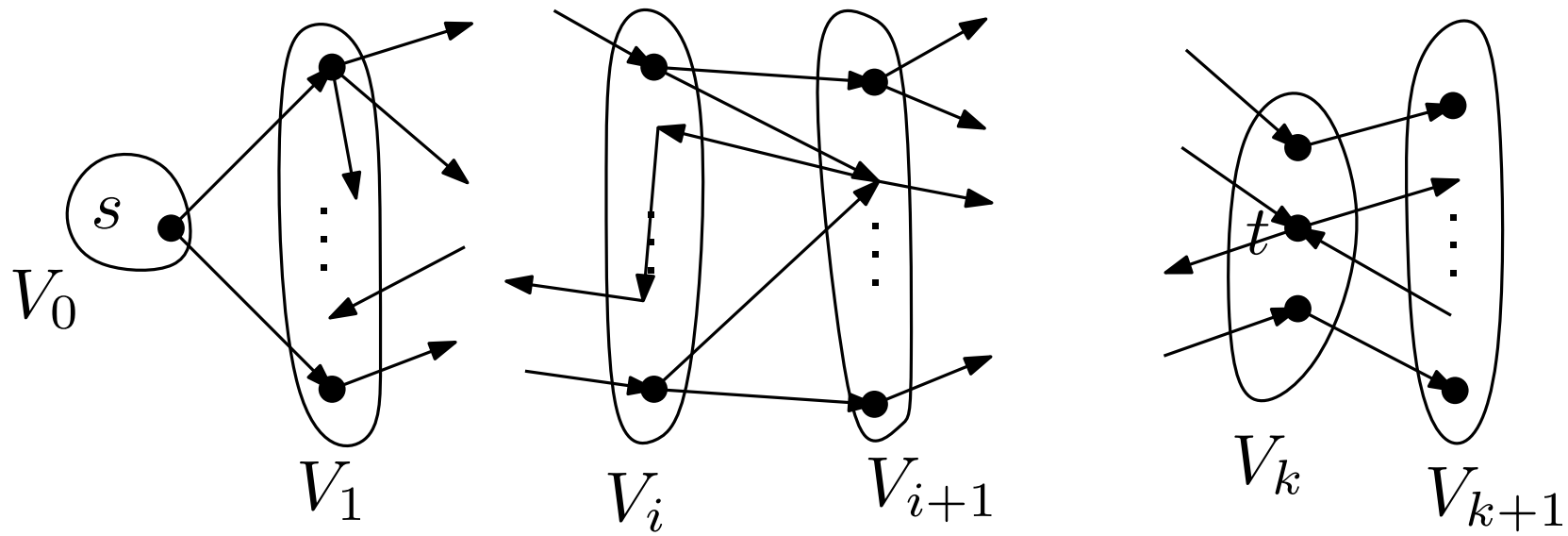
Lemma 2. On a network with unit capacities, Dinic' algorithm performs at most $O\left(n^{2/3}\right)$ phases, and thus $O(n^{2/3}m)$ steps.

Proof. $V_i := \{v \mid \text{dist}(s,v) = i \text{ in } G_t\}$.



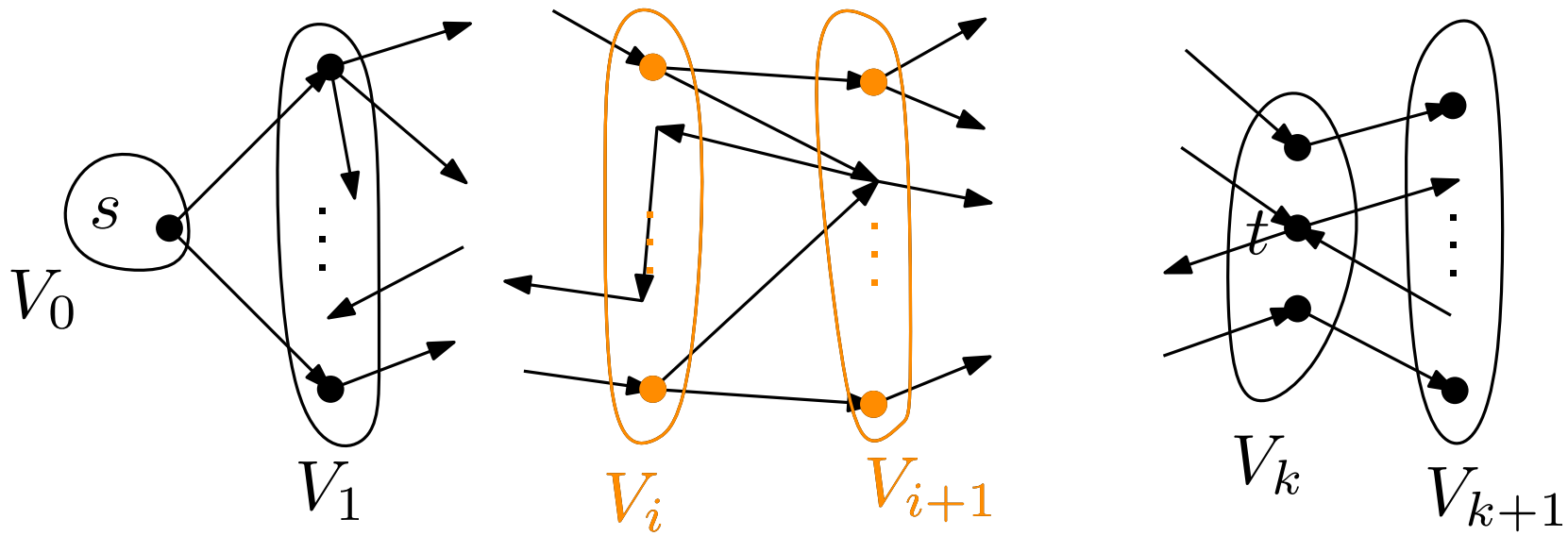
V_i is *big* if $|V_i| > 2n/k$.

Fewer than $\frac{n}{2n/k} = \frac{k}{2}$ sets V_i are big.



V_i is *big* if $|V_i| > 2n/k$.

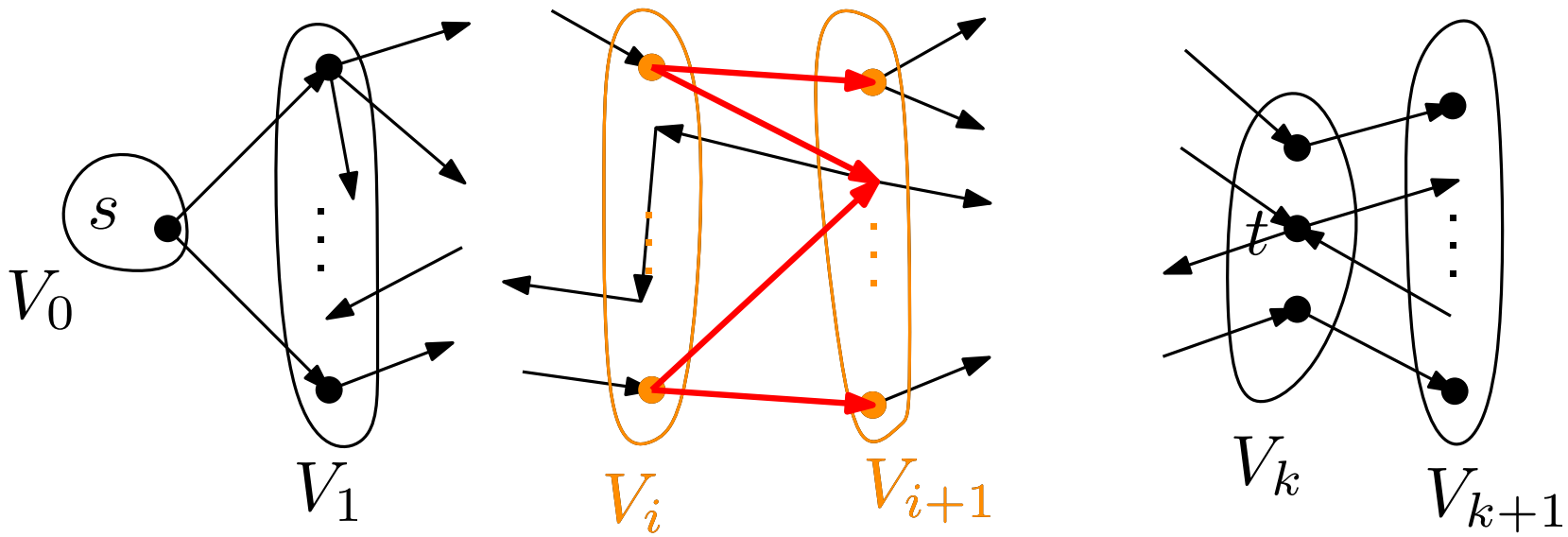
Fewer than $\frac{n}{2n/k} = \frac{k}{2}$ sets V_i are big.



V_i is *big* if $|V_i| > 2n/k$.

Fewer than $\frac{n}{2n/k} = \frac{k}{2}$ sets V_i are big.

Thus, $V_0, \ldots, V_k$ contains two consecutive "small" sets V_i, V_{i+1}.

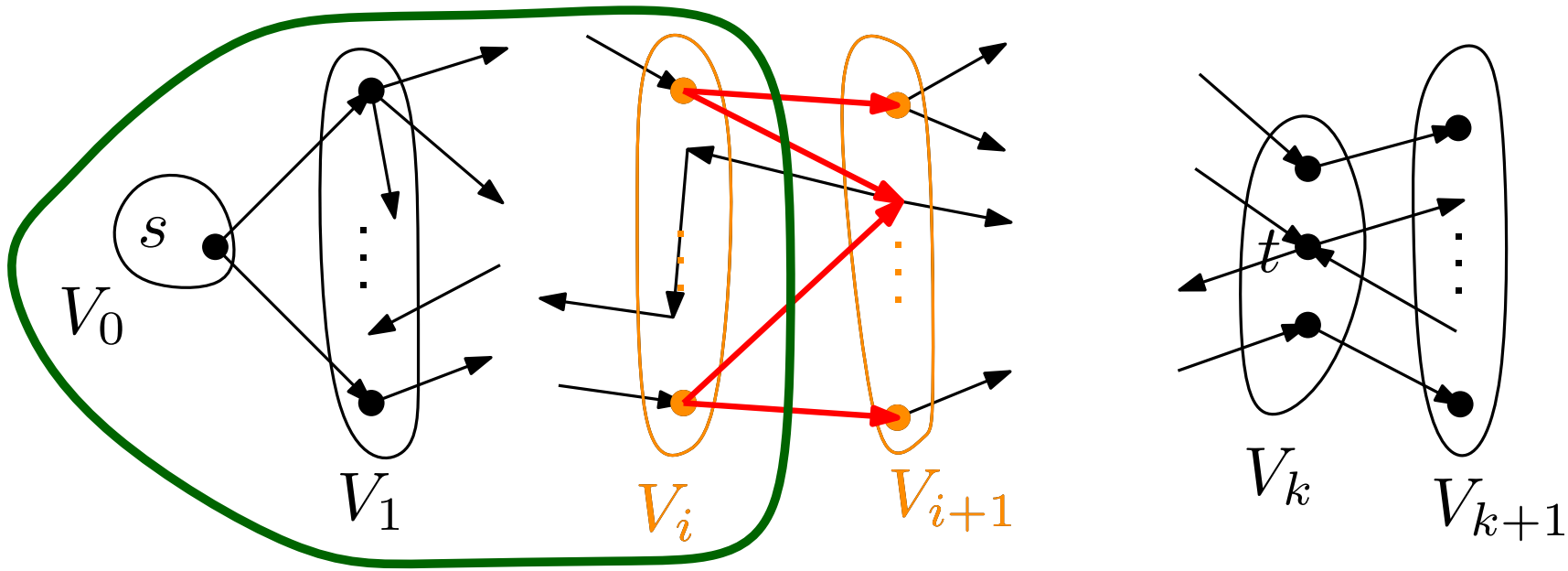


V_i is *big* if $|V_i| > 2n/k$.

Fewer than $\frac{n}{2n/k} = \frac{k}{2}$ sets V_i are big.

Thus, $V_0, \ldots, V_k$ contains two consecutive "small" sets V_i, V_{i+1}.

$|E(V_i, V_{i+1})| \leq |V_i| \cdot |V_{i+1}| \leq \frac{4n^2}{k^2}$.



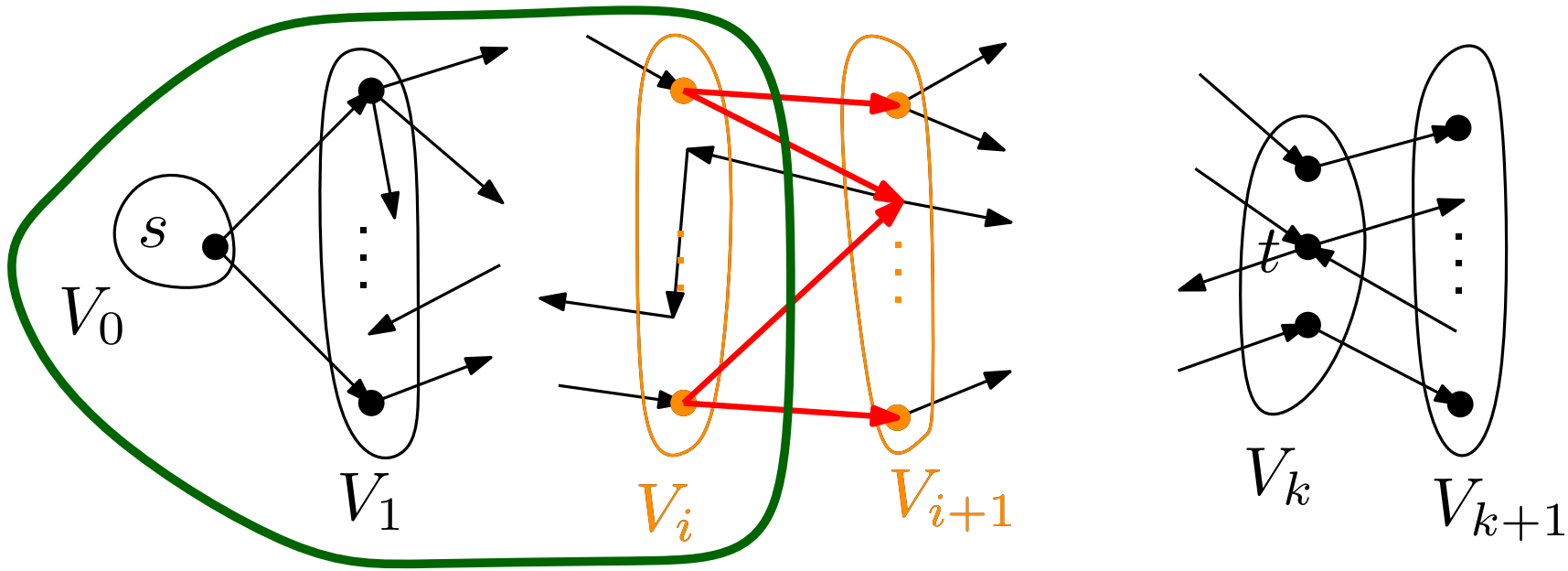
V_i is *big* if $|V_i| > 2n/k$.

Fewer than $\frac{n}{2n/k} = \frac{k}{2}$ sets V_i are big.

Thus, $V_0, \ldots, V_k$ contains two consecutive "small" sets V_i, V_{i+1}.

$|E(V_i, V_{i+1})| \leq |V_i| \cdot |V_{i+1}| \leq \frac{4n^2}{k^2}$.

$S := V_0 \cup V_1 \cup \ldots V_i$ is an s-t-cut.



V_i is *big* if $|V_i| > 2n/k$.

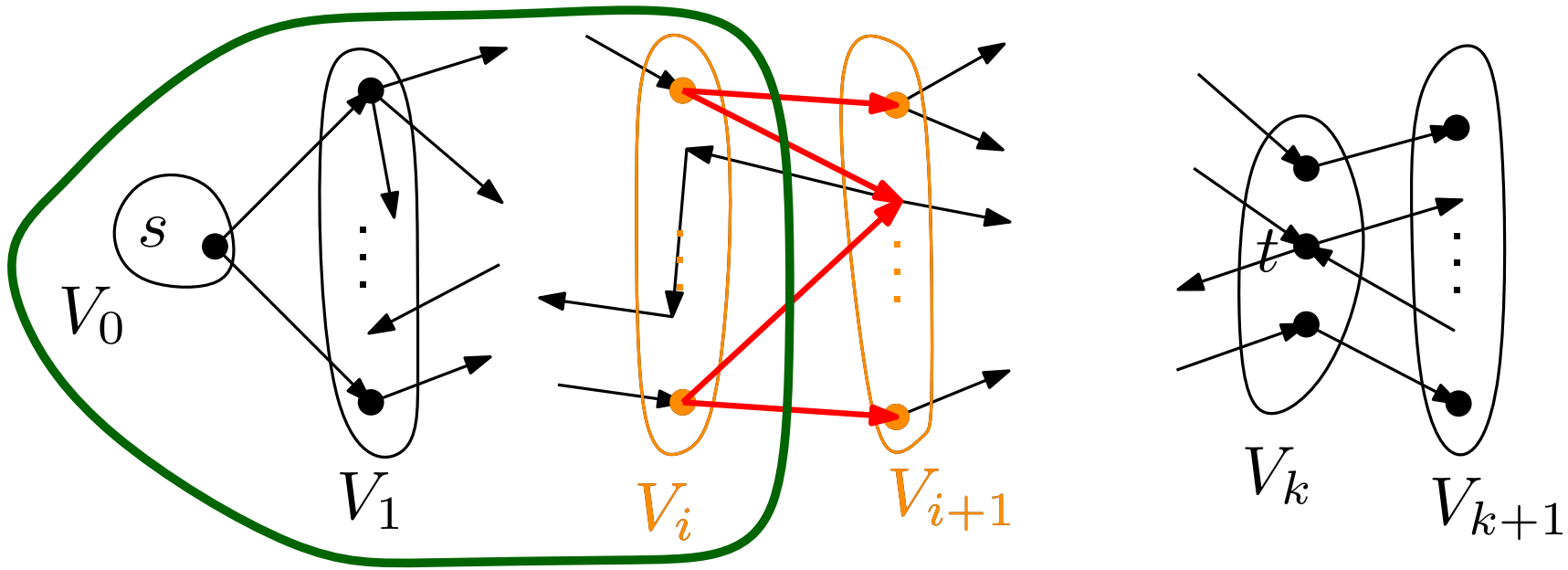
Fewer than $\frac{n}{2n/k} = \frac{k}{2}$ sets V_i are big.

Thus, $V_0, \ldots, V_k$ contains two consecutive "small" sets V_i, V_{i+1}.

$|E(V_i, V_{i+1})| \leq |V_i| \cdot |V_{i+1}| \leq \frac{4n^2}{k^2}$.

$S := V_0 \cup V_1 \cup \ldots V_i$ is an s-t-cut.

$$\text{cap}(S, V \setminus S) \leq \frac{4n^2}{k^2} \leq \frac{4n^2}{t^2}.$$



V_i is *big* if $|V_i| > 2n/k$.

Fewer than $\frac{n}{2n/k} = \frac{k}{2}$ sets V_i are big.

Thus, $V_0, \ldots, V_k$ contains two consecutive "small" sets V_i, V_{i+1}.

$|E(V_i, V_{i+1})| \leq |V_i| \cdot |V_{i+1}| \leq \frac{4n^2}{k^2}$.

$S := V_0 \cup V_1 \cup \ldots V_i$ is an s-t-cut.

$$\text{maxflow}(G_t) \leq \text{cap}(S, V \setminus S) \leq \frac{4n^2}{k^2} \leq \frac{4n^2}{t^2}.$$

$G_t :=$ the residual network after t phases.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{4n^2}{t^2}$.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{4n^2}{t^2}$.

Every phase increases the flow by at least 1.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{4n^2}{t^2}$.

Every phase increases the flow by at least 1.

At most $t + \frac{4n^2}{t^2}$ phases.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{4n^2}{t^2}$.

Every phase increases the flow by at least 1.

At most $t + \frac{4n^2}{t^2}$ phases.

At most $5n^{2/3}$ phases (setting $t = n^{2/3}$).

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{4n^2}{t^2}$.

Every phase increases the flow by at least 1.

At most $t + \frac{4n^2}{t^2}$ phases.

At most $5n^{2/3}$ phases (setting $t = n^{2/3}$).

Lemma 2. On a network with unit capacities, Dinic' algorithm performs at most $O\left(n^{2/3}\right)$ phases, and thus $O(n^{2/3}m)$ steps.

$G_t :=$ the residual network after t phases.

maxflow$(G_t) \leq \frac{4n^2}{t^2}$.

Every phase increases the flow by at least 1.

At most $t + \frac{4n^2}{t^2}$ phases.

At most $5n^{2/3}$ phases (setting $t = n^{2/3}$).

Lemma 2. On a network with unit capacities, Dinic' algorithm performs at most $O\left(n^{2/3}\right)$ phases, and thus $O(n^{2/3}m)$ steps.

■

Theorem. Dinic' algorithm finds a maximum flow in $O(n^2m)$ steps.

Theorem. On a network with unit capacities, Dinic' algorithm finds a maximum flow in $O(m^{1.5})$ steps.

Theorem. On a network with unit capacities, Dinic' algorithm finds a maximum flow in $O(n^{2/3}m)$ steps.