Preflows

Definition 64

An (s,t)-preflow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

Preflows

Definition 64

An (s,t)-preflow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

1. For each edge e

$$0 \leq f(e) \leq c(e) \ .$$

(capacity constraints)

2. For each $v \in V \setminus \{s,t\}$

$$\sum_{e \in \text{out}(v)} f(e) \leq \sum_{e \in \text{into}(v)} f(e) \ .$$

Preflows

Definition 64

An (s,t)-preflow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

1. For each edge e

$$0 \leq f(e) \leq c(e) \ .$$

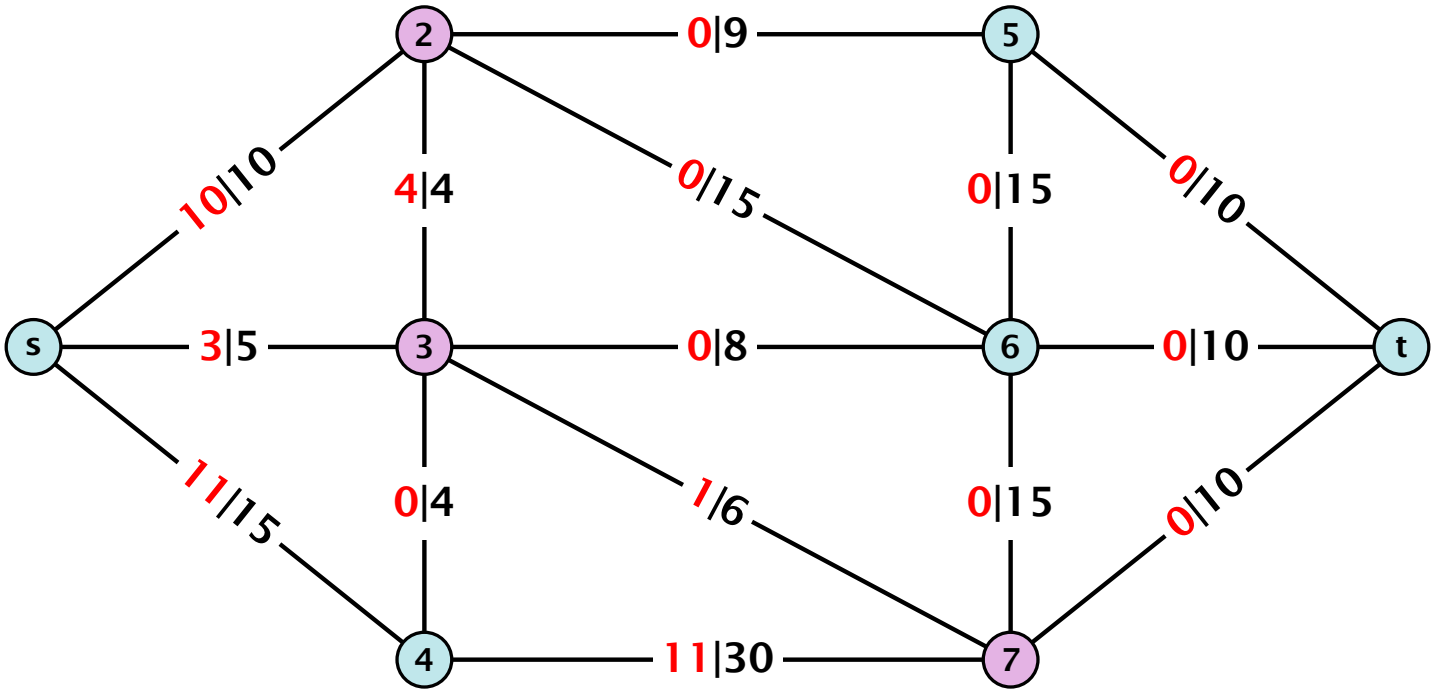
(capacity constraints)

2. For each $v \in V \setminus \{s,t\}$

$$\sum_{e \in \text{out}(v)} f(e) \leq \sum_{e \in \text{into}(v)} f(e) \ .$$

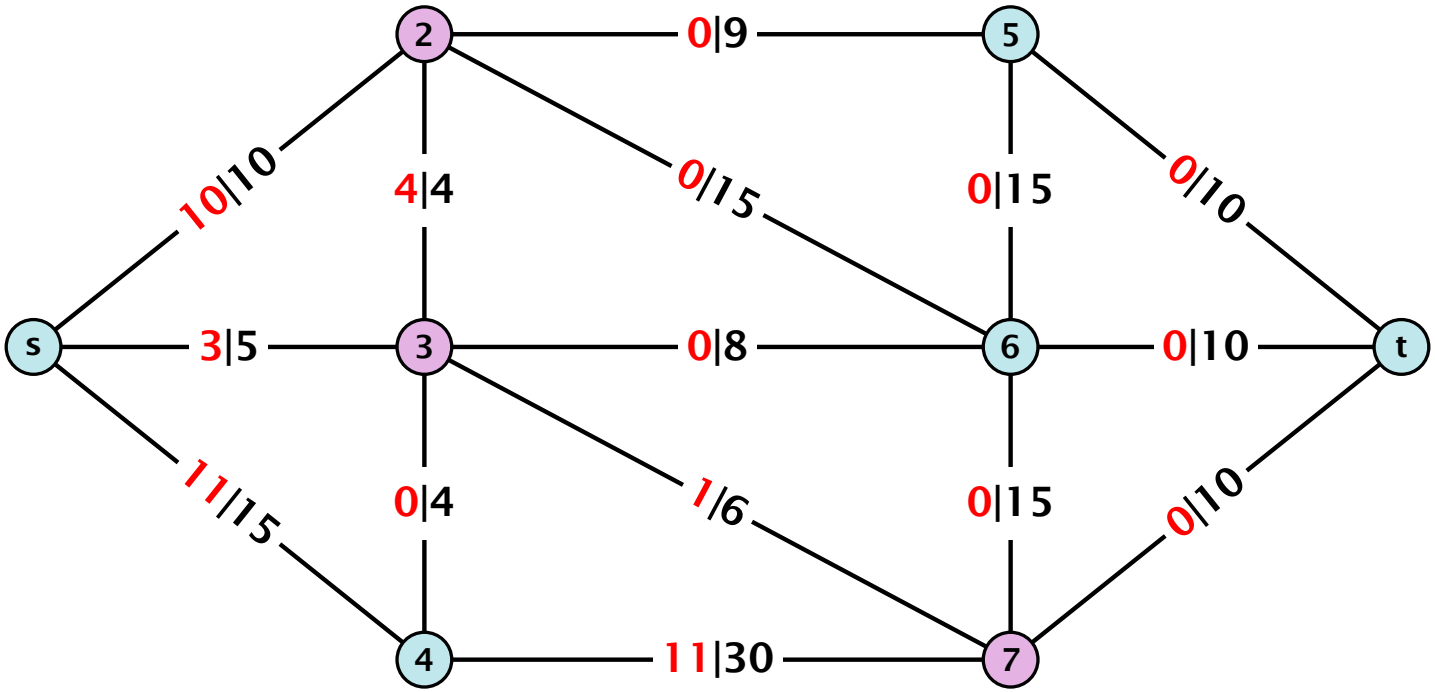
Preflows

Example 65



Preflows

Example 65



A node that has $\sum_{e \in \text{out}(v)} f(e) < \sum_{e \in \text{into}(v)} f(e)$ is called an active node.

Preflows

Preflows

Definition:

A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \leq \ell(v) + 1$ for all edges in the residual graph G_f (only non-zero capacity edges!!!)

Preflows

Definition:

A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \leq \ell(v) + 1$ for all edges in the residual graph G_f (only non-zero capacity edges!!!)
- $\ell(s) = n$

Preflows

Definition:

A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \leq \ell(v) + 1$ for all edges in the residual graph G_f (only non-zero capacity edges!!!)
- $\ell(s) = n$
- $\ell(t) = 0$

Preflows

Definition:

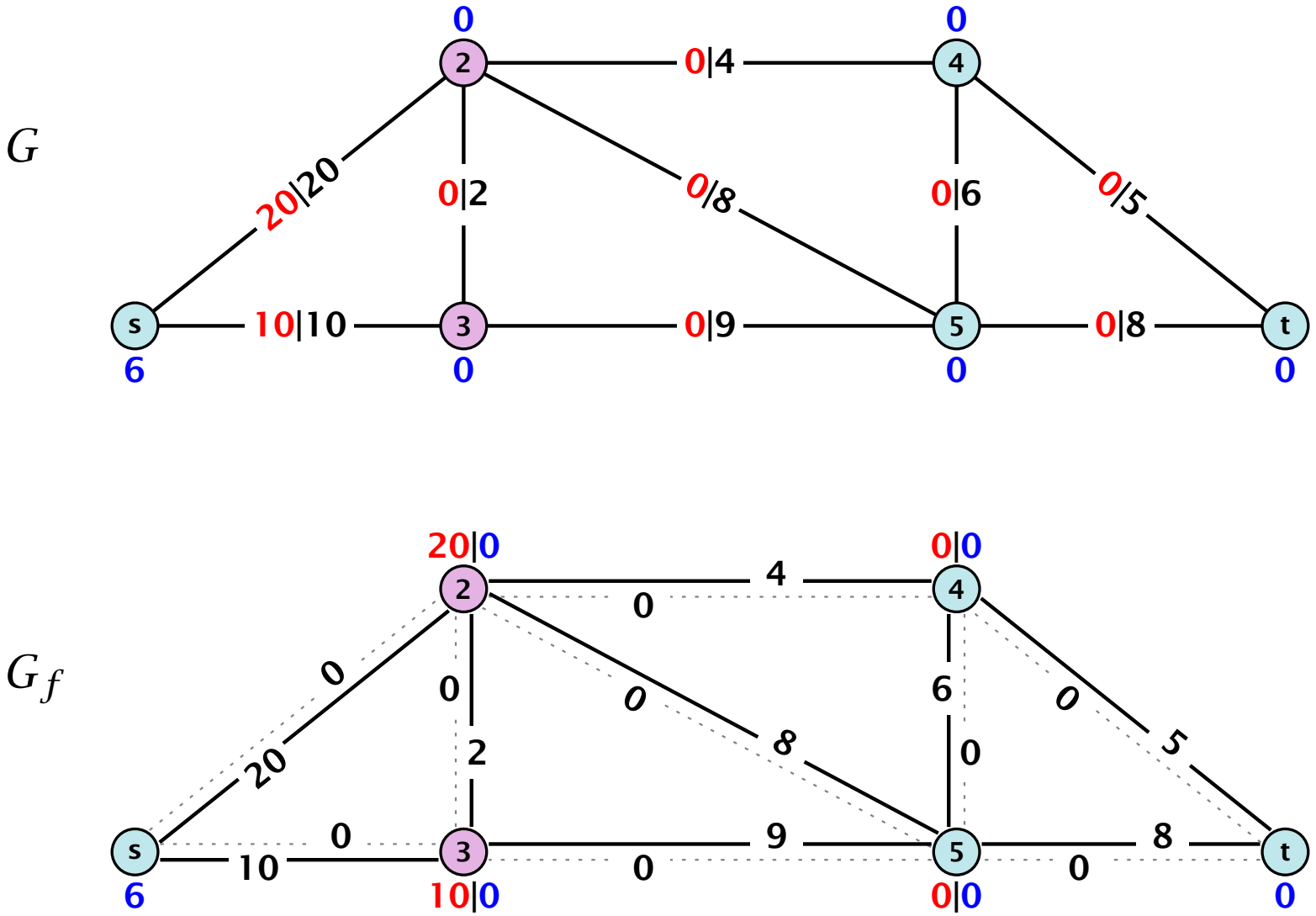
A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \leq \ell(v) + 1$ for all edges in the residual graph G_f (only non-zero capacity edges!!!)
- $\ell(s) = n$
- $\ell(t) = 0$

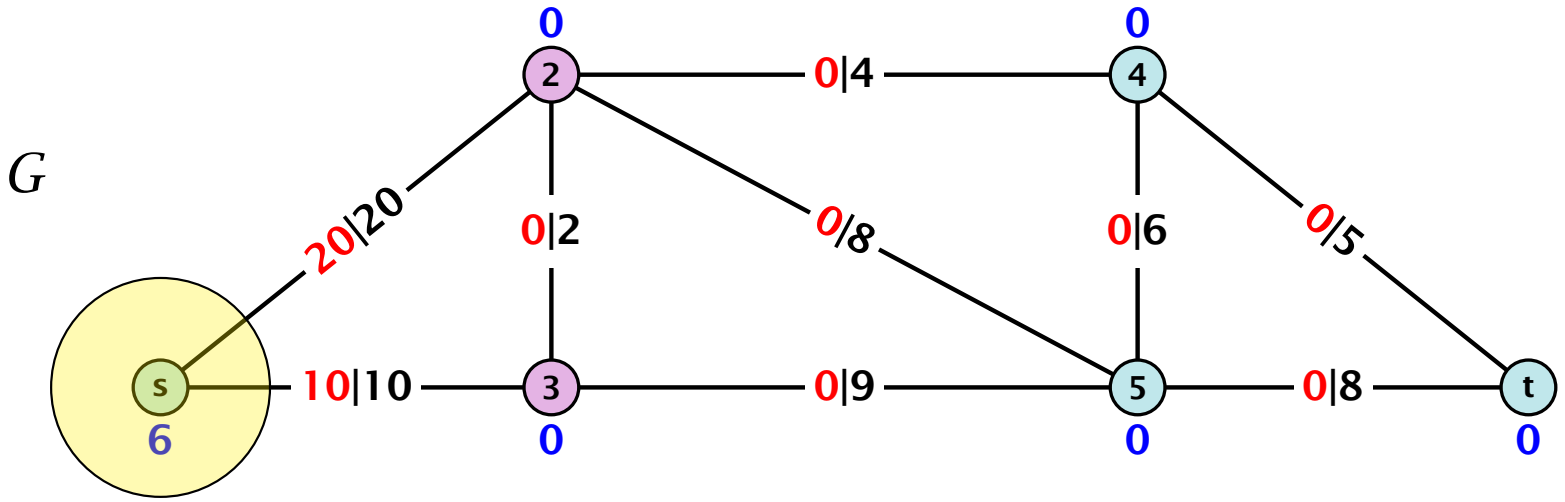
Intuition:

The labelling can be viewed as a height function. Whenever the height from node u to node v decreases by more than 1 (i.e., it goes very steep downhill from u to v), the corresponding edge must be saturated.

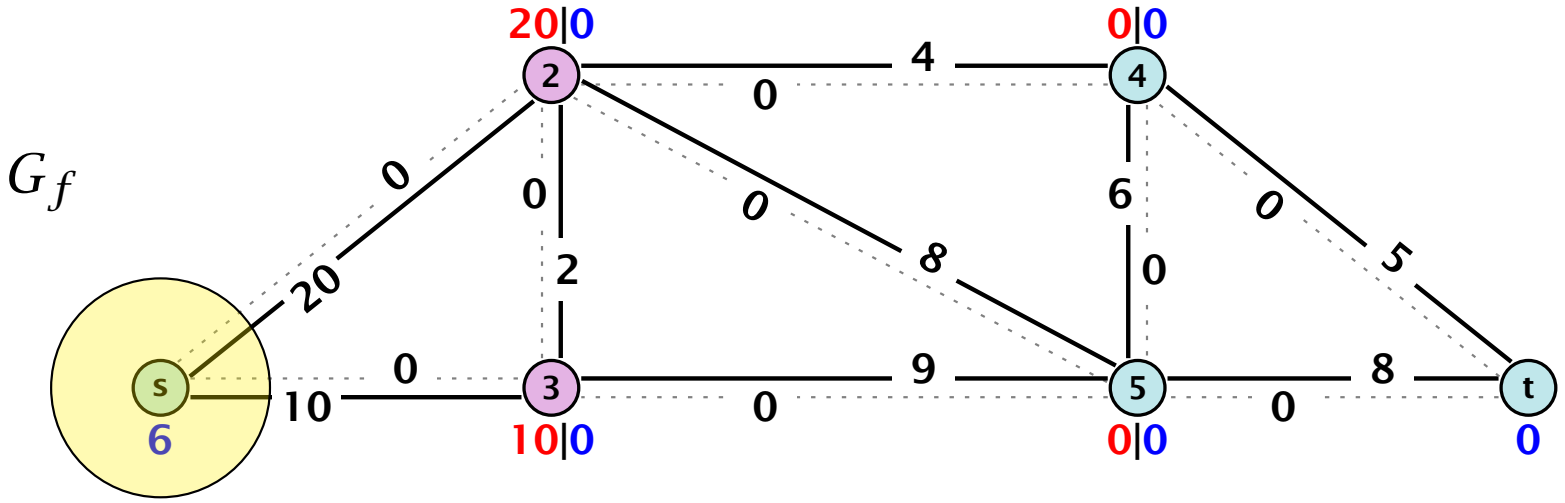
Preflows


G
0
0
2
4
0|4
20|20
0|2
0|8
0|6
0|5
s
10|10
3
0|9
5
0|8
t
6
0
0
0
G_f
20|0
0|0
2
4
4
0
0
0
6
0
20
2
8
5
s
0
3
9
5
8
t
10
0
0
0
6
10|0
0|0
0

Preflows

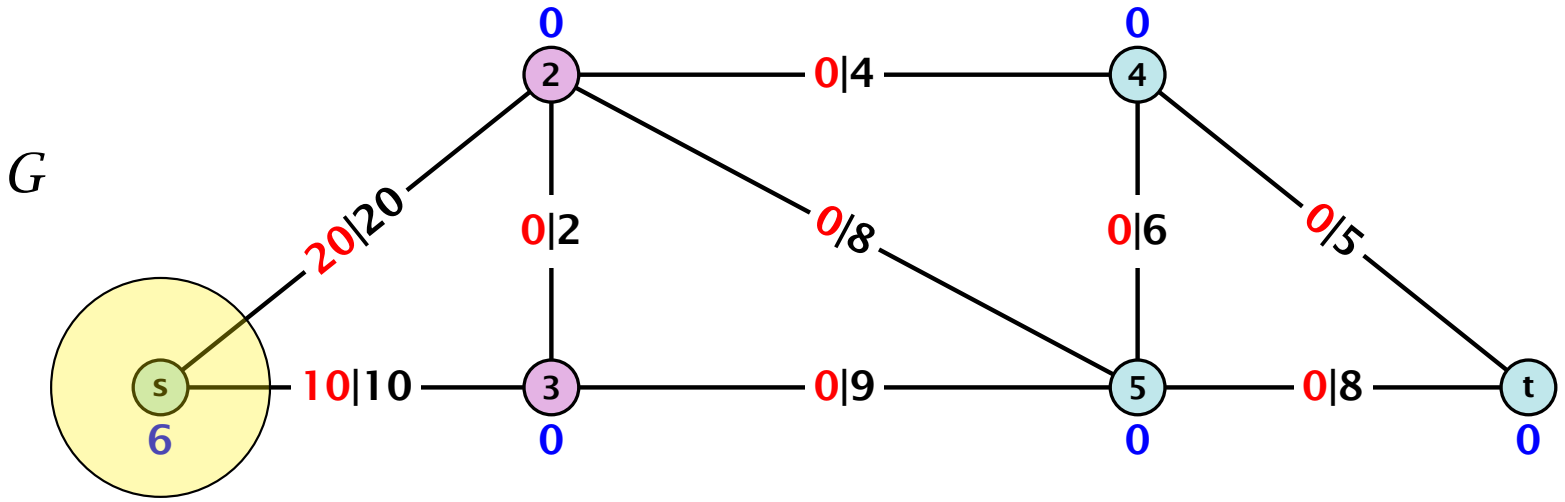


G
0
0
2
4
0|4
20|20
0|2
0|8
0|6
0|5
s
10|10
3
0|9
5
0|8
t
6
0
0
0

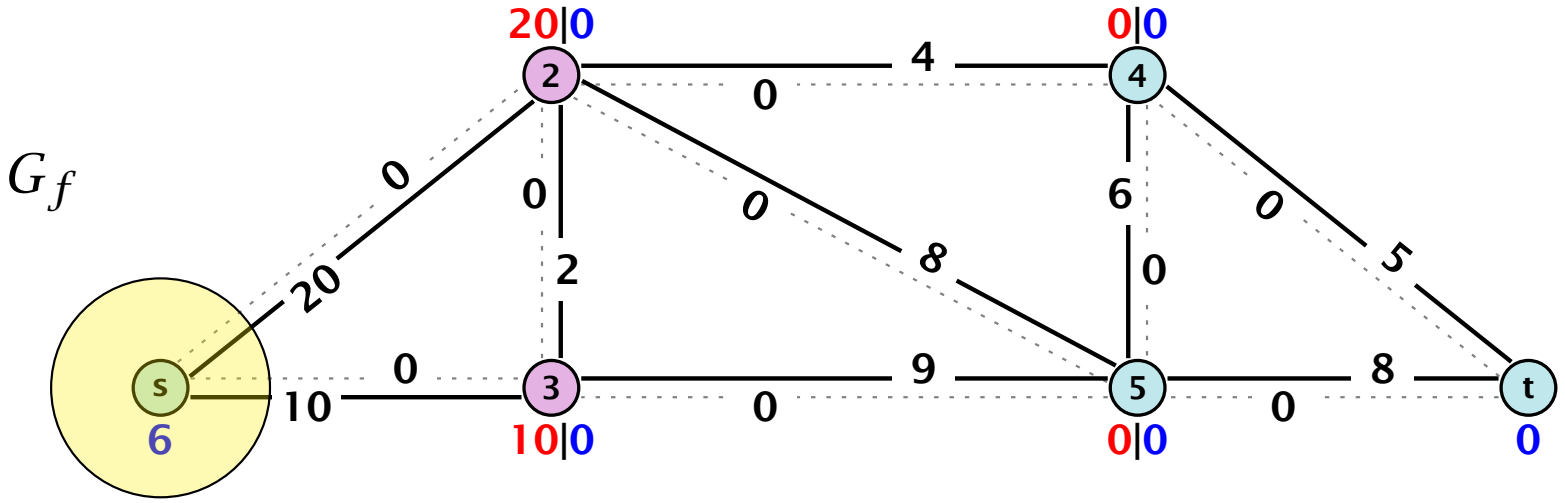


G_f
20|0
0|0
2
4
4
0
0
0
0
6
0
20
2
8
0
5
s
0
3
9
5
8
t
10
0
0
6
10|0
0|0
0

Preflows



G
0
0
2
4
0|4
20|20
0|2
0|8
0|6
0|5
s
10|10
3
0|9
5
0|8
t
6
0
0
0



G_f
20|0
0|0
2
4
4
0
0
0
0
6
0
20
2
8
0
5
s
0
3
9
5
8
t
10
0
0
6
10|0
0|0
0

Preflows

Preflows

Lemma 66

A preflow that has a valid labelling saturates a cut.

Preflows

Lemma 66

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0, \ldots, n$.

Preflows

Lemma 66

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0, \ldots, n$.
- There must exist a label $d \in \{0, \ldots, n\}$ such that none of the nodes carries this label.

Preflows

Lemma 66

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0,\ldots,n$.
- There must exist a label $d \in \{0,\ldots,n\}$ such that none of the nodes carries this label.
- Let $A = \{v \in V \mid \ell(v) > d\}$ and $B = \{v \in V \mid \ell(v) < d\}$.

Preflows

Lemma 66

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0,\ldots,n$.
- There must exist a label $d \in \{0,\ldots,n\}$ such that none of the nodes carries this label.
- Let $A = \{v \in V \mid \ell(v) > d\}$ and $B = \{v \in V \mid \ell(v) < d\}$.
- We have $s \in A$ and $t \in B$ and there is no edge from A to B in the residual graph G_f; this means that (A,B) is a saturated cut.

Preflows

Lemma 66

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0,\ldots,n$.
- There must exist a label $d \in \{0,\ldots,n\}$ such that none of the nodes carries this label.
- Let $A = \{v \in V \mid \ell(v) > d\}$ and $B = \{v \in V \mid \ell(v) < d\}$.
- We have $s \in A$ and $t \in B$ and there is no edge from A to B in the residual graph G_f; this means that (A,B) is a saturated cut.

Lemma 67

A flow that has a valid labelling is a maximum flow.

Push Relabel Algorithms

Push Relabel Algorithms

Idea:

- start with some preflow and some valid labelling

Push Relabel Algorithms

Idea:

- start with some preflow and some valid labelling
- successively change the preflow while maintaining a valid labelling

Push Relabel Algorithms

Idea:

- start with some preflow and some valid labelling
- successively change the preflow while maintaining a valid labelling
- stop when you have a flow (i.e., no more active nodes)

Changing a Preflow

An arc (u,v) with $c_f(u,v) > 0$ in the residual graph is admissable if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \text{into}(u)} f(e) - \sum_{e \in \text{out}(u)} f(e)$ and suppose $e = (u,v)$ is an admissable arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

Changing a Preflow

An arc (u,v) with $c_f(u,v) > 0$ in the residual graph is admissable if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \text{into}(u)} f(e) - \sum_{e \in \text{out}(u)} f(e)$ and suppose $e = (u,v)$ is an admissable arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

Changing a Preflow

An arc (u,v) with $c_f(u,v) > 0$ in the residual graph is admissable if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \text{into}(u)} f(e) - \sum_{e \in \text{out}(u)} f(e)$ and suppose $e = (u,v)$ is an admissable arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

Changing a Preflow

An arc (u,v) with $c_f(u,v) > 0$ in the residual graph is admissable if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \text{into}(u)} f(e) - \sum_{e \in \text{out}(u)} f(e)$ and suppose $e = (u,v)$ is an admissable arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

Changing a Preflow

An arc (u,v) with $c_f(u,v) > 0$ in the residual graph is admissable if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \text{into}(u)} f(e) - \sum_{e \in \text{out}(u)} f(e)$ and suppose $e = (u,v)$ is an admissable arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

- saturating push: $\min\{f(u), c_f(e)\} = c_f(e)$
  the arc e is deleted from the residual graph
- non-saturating push: $\min\{f(u), c_f(e)\} = f(u)$
  the node u becomes inactive

Changing a Preflow

An arc (u,v) with $c_f(u,v) > 0$ in the residual graph is admissable if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \text{into}(u)} f(e) - \sum_{e \in \text{out}(u)} f(e)$ and suppose $e = (u,v)$ is an admissable arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

- saturating push: $\min\{f(u), c_f(e)\} = c_f(e)$
  the arc e is deleted from the residual graph
- non-saturating push: $\min\{f(u), c_f(e)\} = f(u)$
  the node u becomes inactive

Push Relabel Algorithms

Push Relabel Algorithms

The relabel operation

Consider an active node u that does not have an outgoing admissable arc.

Push Relabel Algorithms

The relabel operation

Consider an active node u that does not have an outgoing admissable arc.

Increasing the label of u by 1 results in a valid labelling.

Push Relabel Algorithms

The relabel operation

Consider an active node u that does not have an outgoing admissable arc.

Increasing the label of u by 1 results in a valid labelling.

- Edges (w,u) incoming to u still fulfill their constraint $\ell(w) \leq \ell(u)+1$.

Push Relabel Algorithms

The relabel operation

Consider an active node u that does not have an outgoing admissable arc.

Increasing the label of u by 1 results in a valid labelling.

- Edges (w,u) incoming to u still fulfill their constraint $\ell(w) \leq \ell(u)+1$.
- An outgoing edge (u,w) had $\ell(u) < \ell(w)+1$ before since it was not admissable. Now: $\ell(u) \leq \ell(w)+1$.

Push Relabel Algorithms

Intuition:
We want to send flow downwards, since the source has a height/label of n and the target a height/label of 0. If we see an active node u with an admissible arc we push the flow at u towards the other end-point that has a lower height/label. If we do not have an admissible arc but excess flow into u it should roughly mean that the level/height/label of u should rise. (If we consider the flow to be water than this would be natural).

Note that the above intuition is very incorrect as the labels are integral, i.e., they cannot really be seen as the height of a node.

Push Relabel Algorithms

Algorithm 47 maxflow(G, s, t, c)

```
1: find initial preflow f
2: while there is active node u do
3:       if there is admiss. arc e out of u then
4:             push(G, e, f, c)
5:       else
6:             relabel(u)
7: return f
```

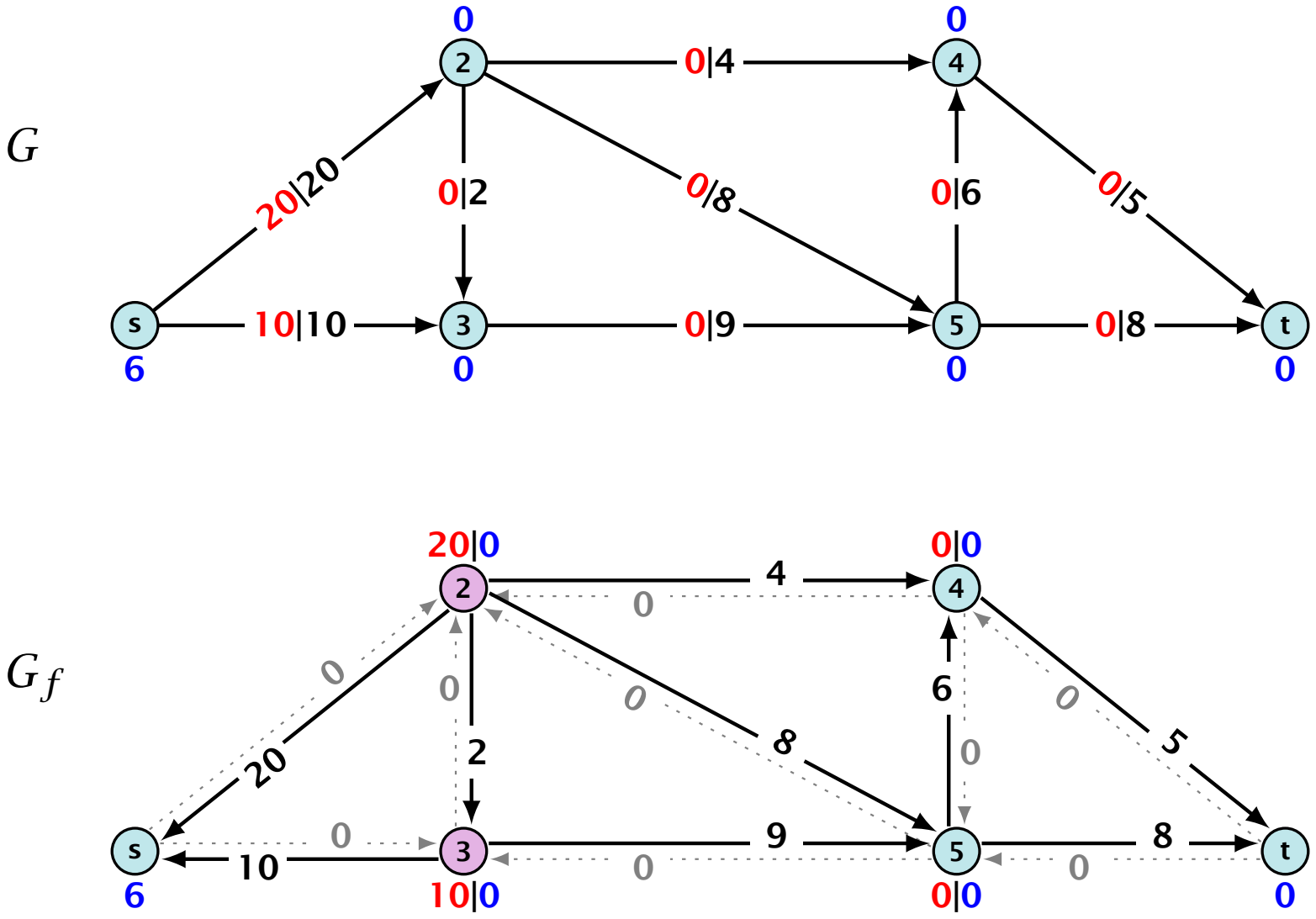
Push Relabel Algorithms

Algorithm 47 maxflow(G, s, t, c)

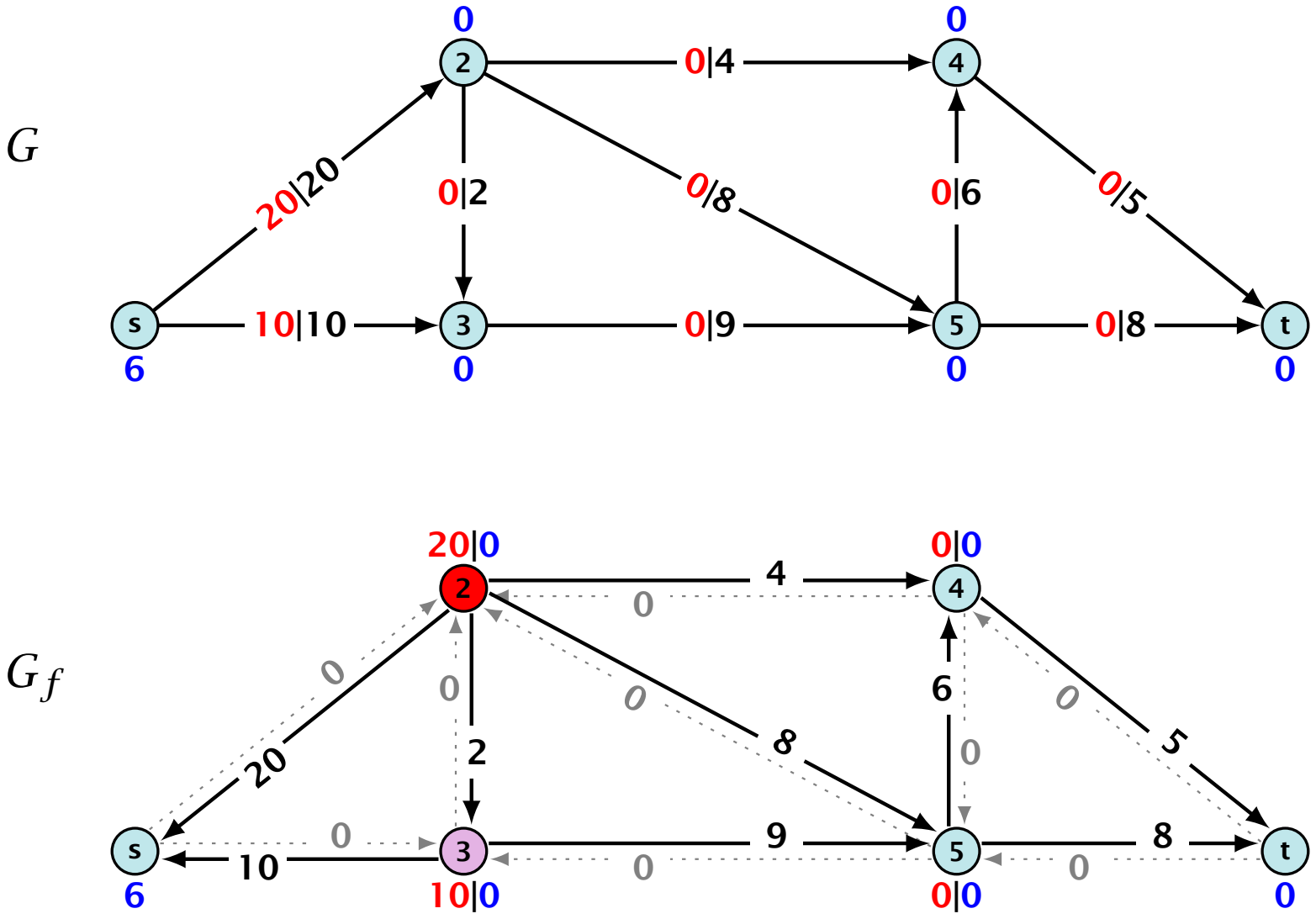
```
1: find initial preflow f
2: while there is active node u do
3:       if there is admiss. arc e out of u then
4:             push(G, e, f, c)
5:       else
6:             relabel(u)
7: return f
```

In the following example we always stick to the same active node u until it becomes inactive but this is not required.

Preflow Push Algorithm



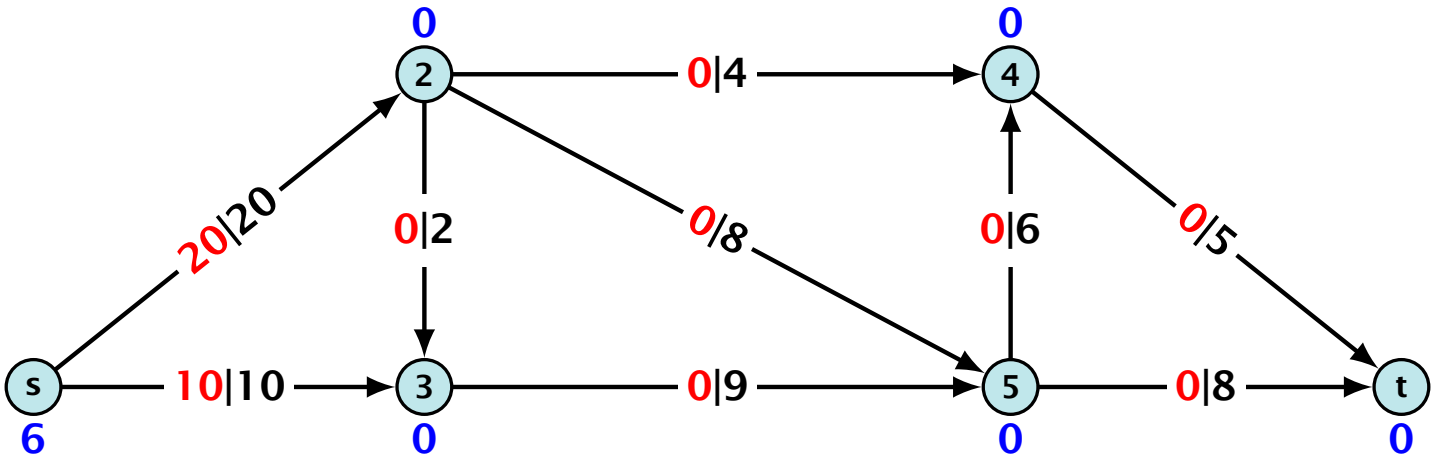
Preflow Push Algorithm



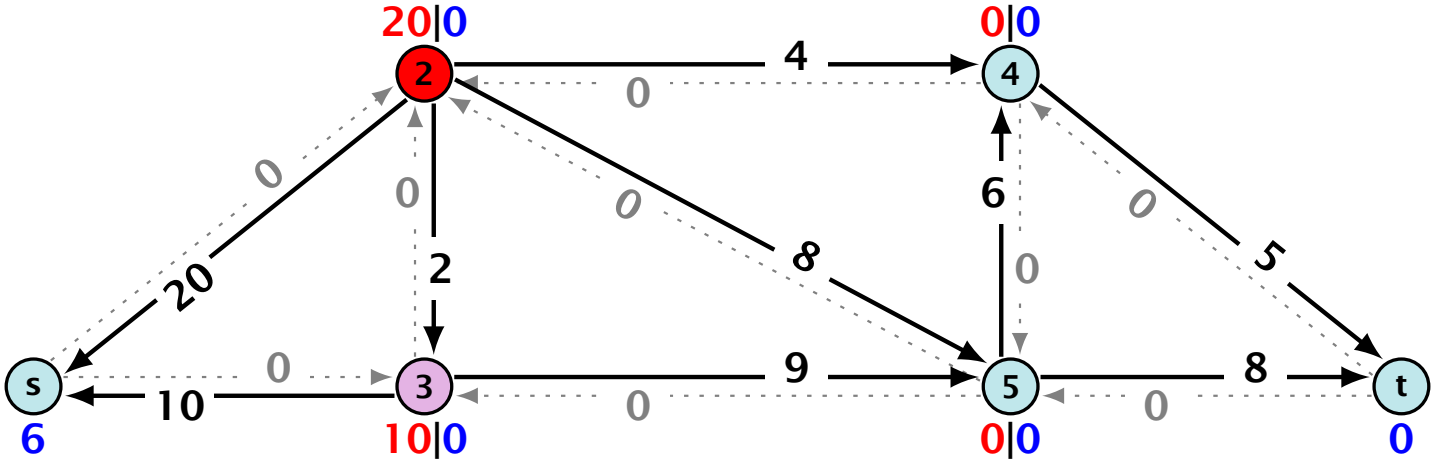
Preflow Push Algorithm

relabel

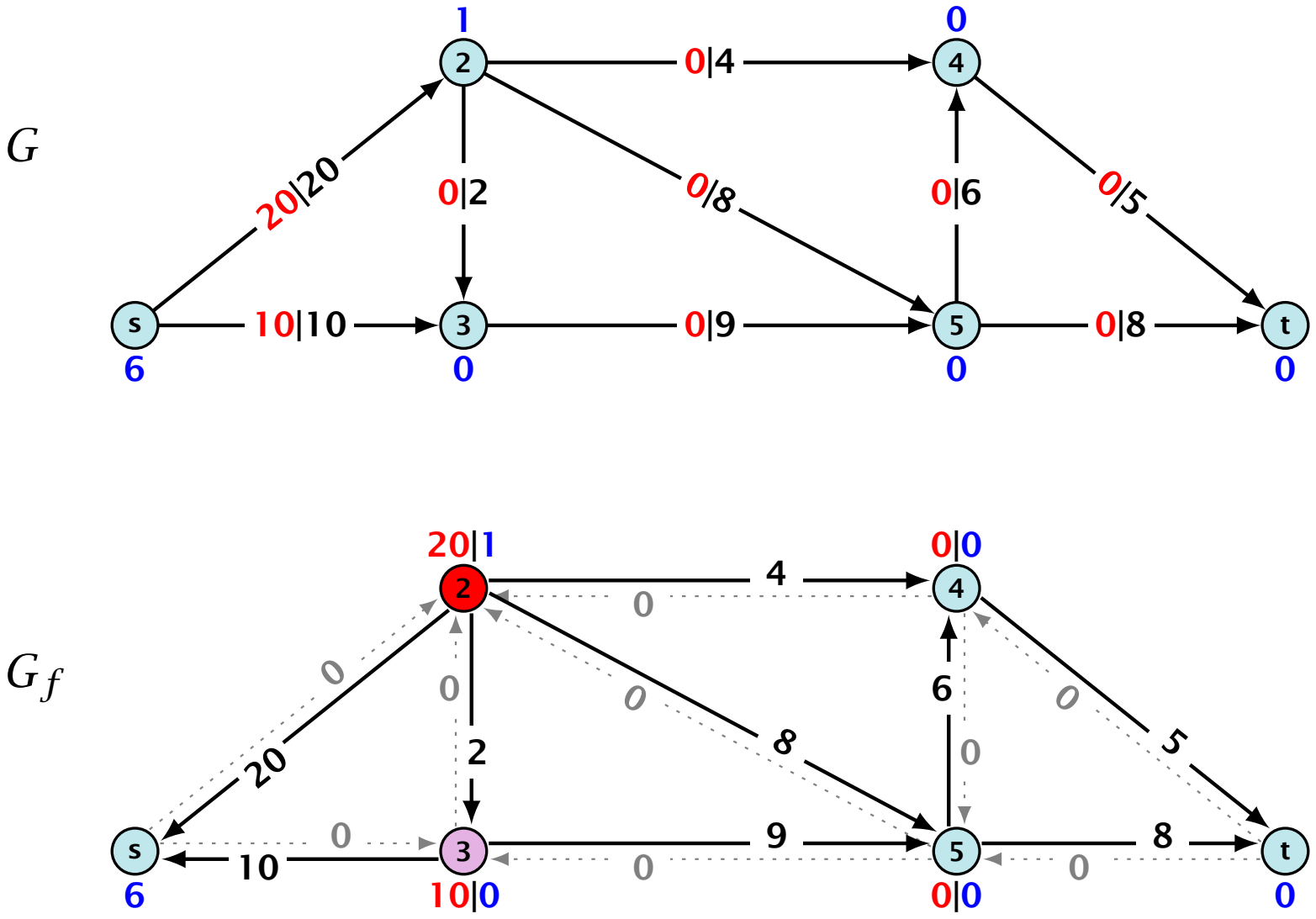
G



G_f

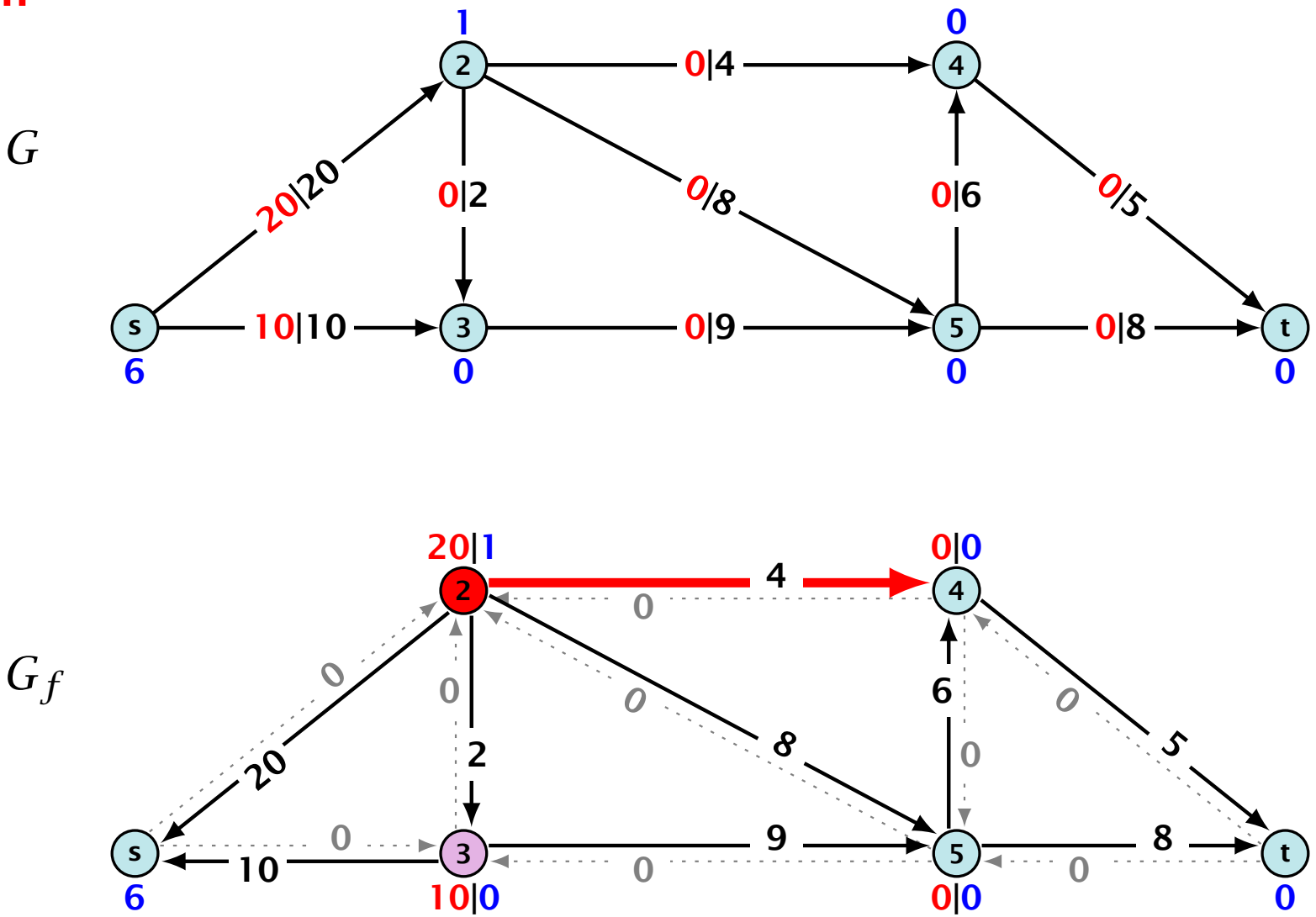


Preflow Push Algorithm

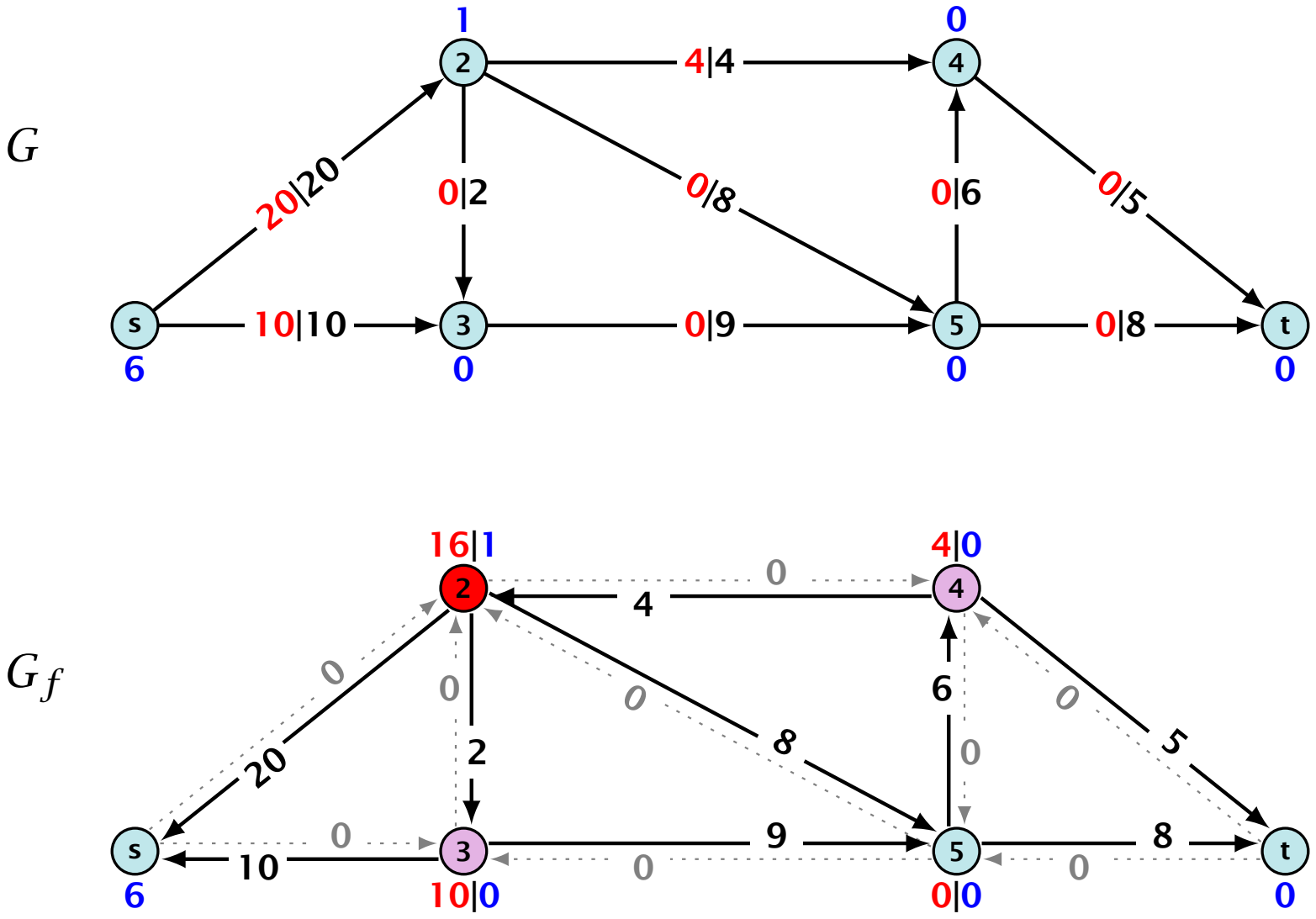


Preflow Push Algorithm

push



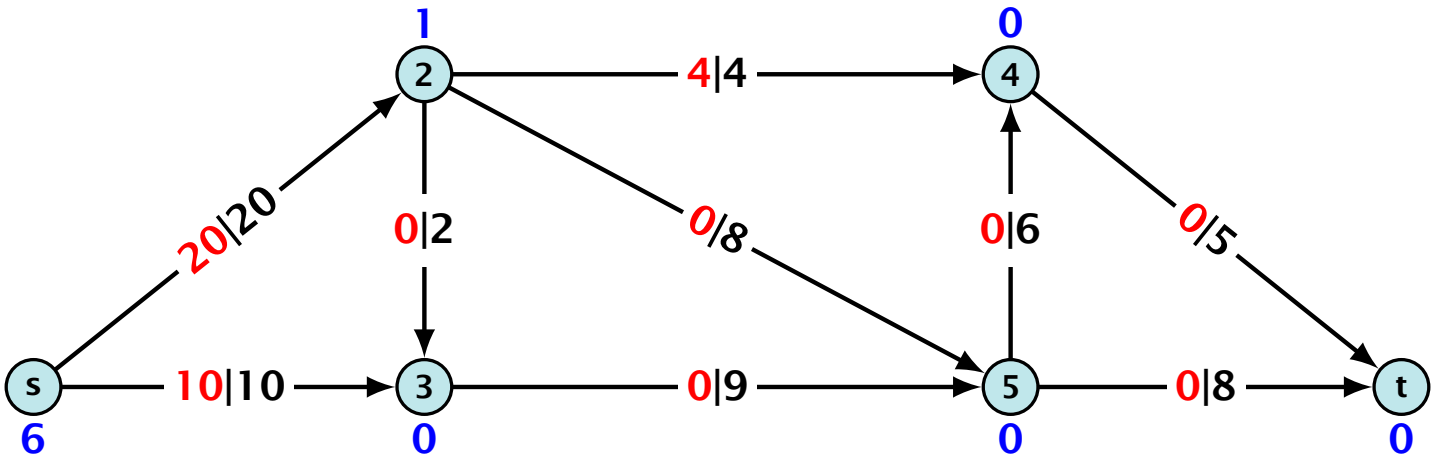
Preflow Push Algorithm



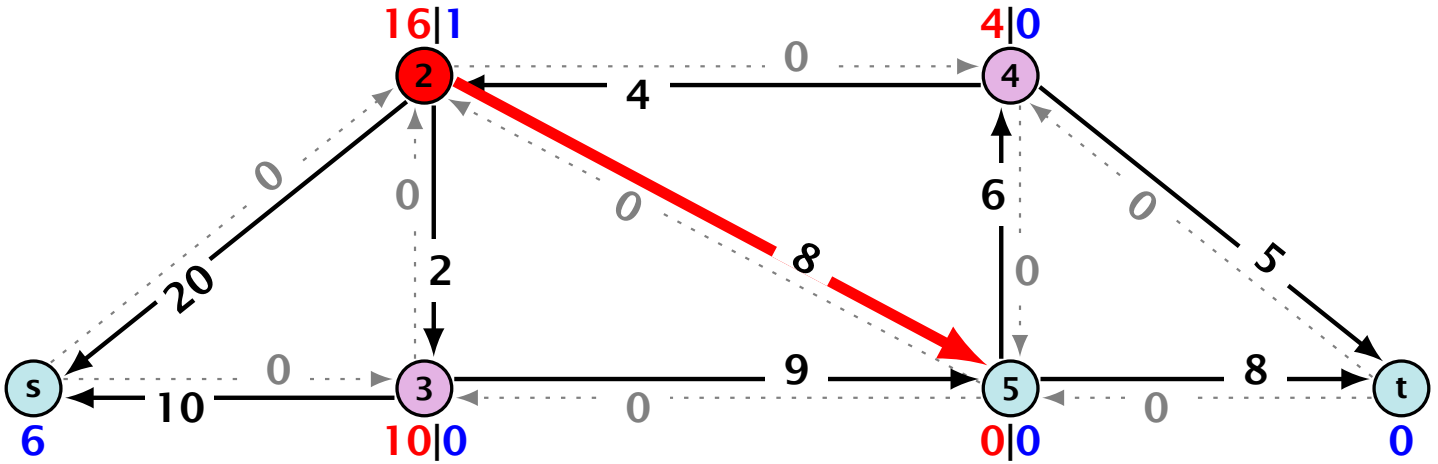
Preflow Push Algorithm

push

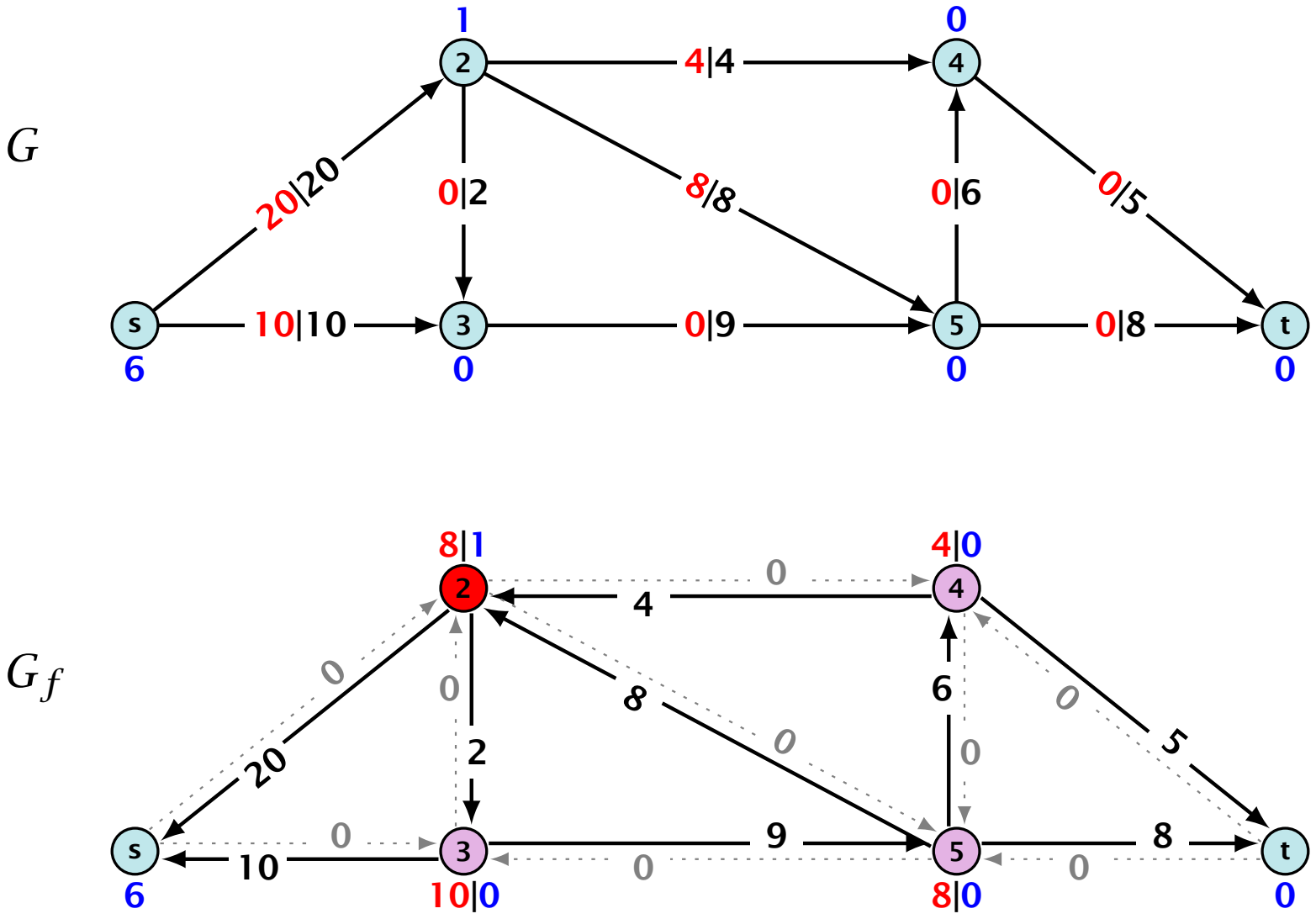
G



G_f

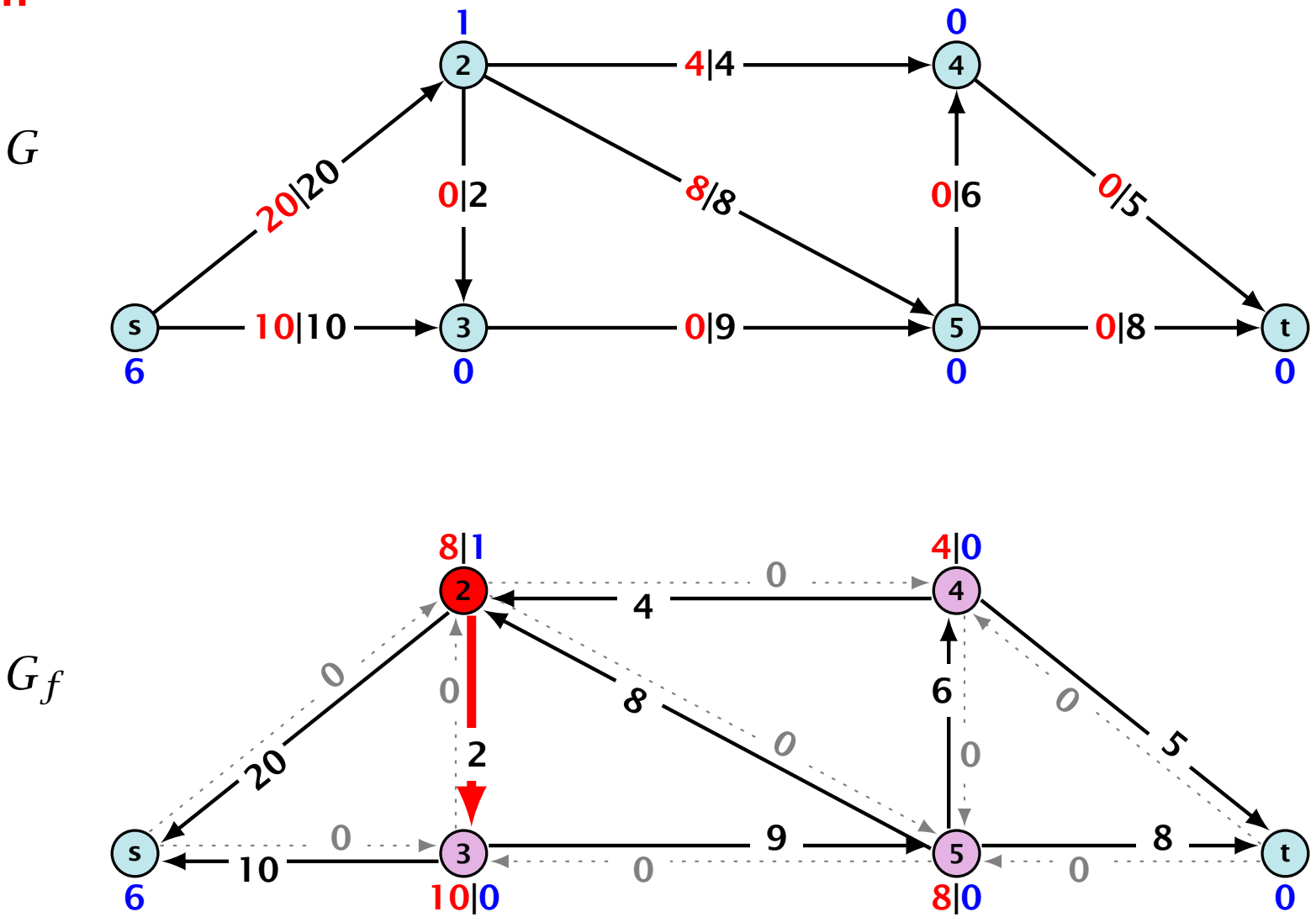


Preflow Push Algorithm

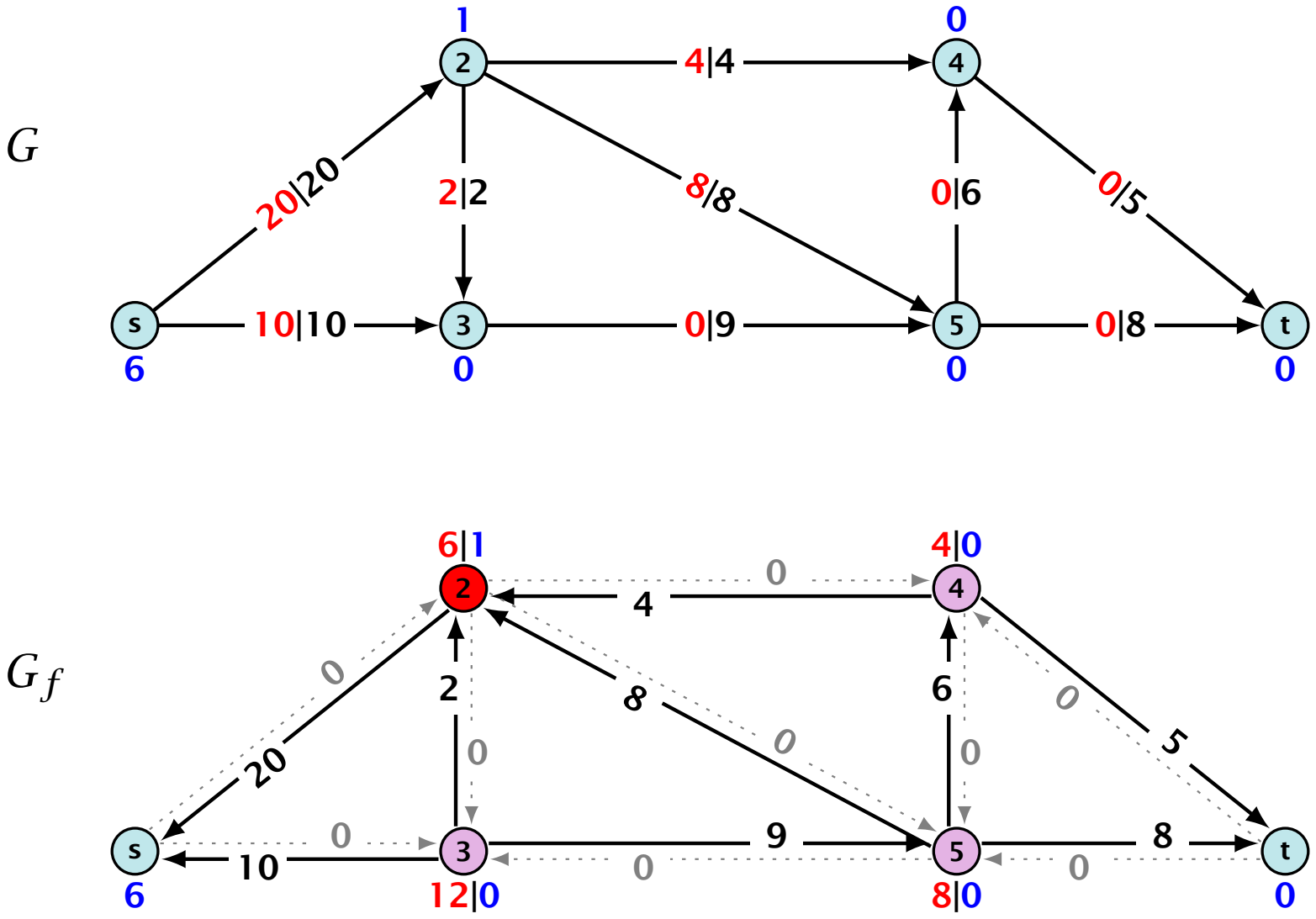


Preflow Push Algorithm

push



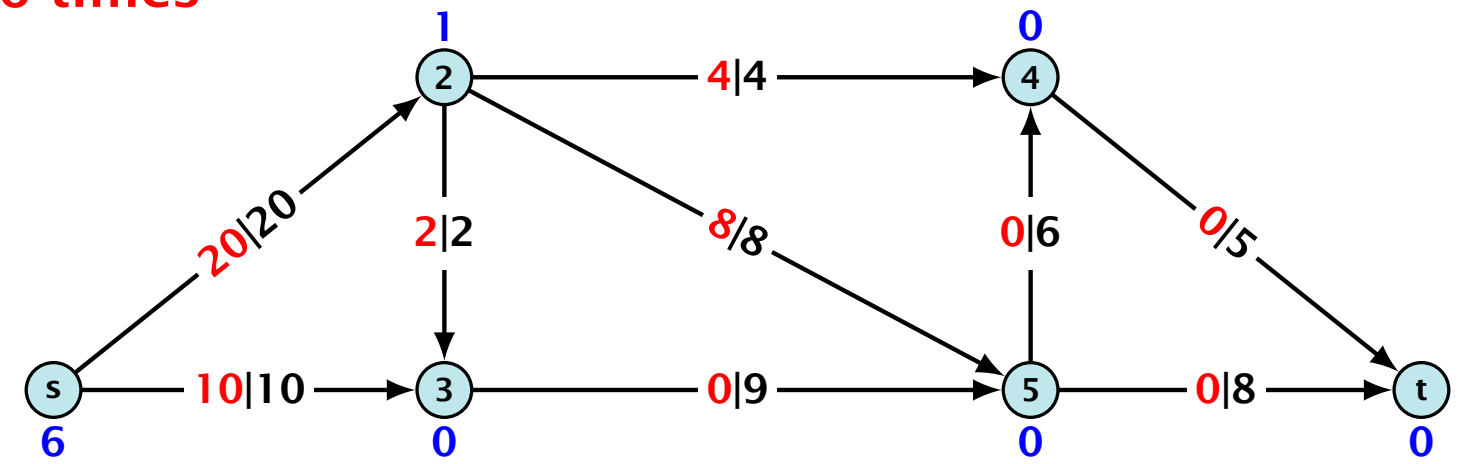
Preflow Push Algorithm



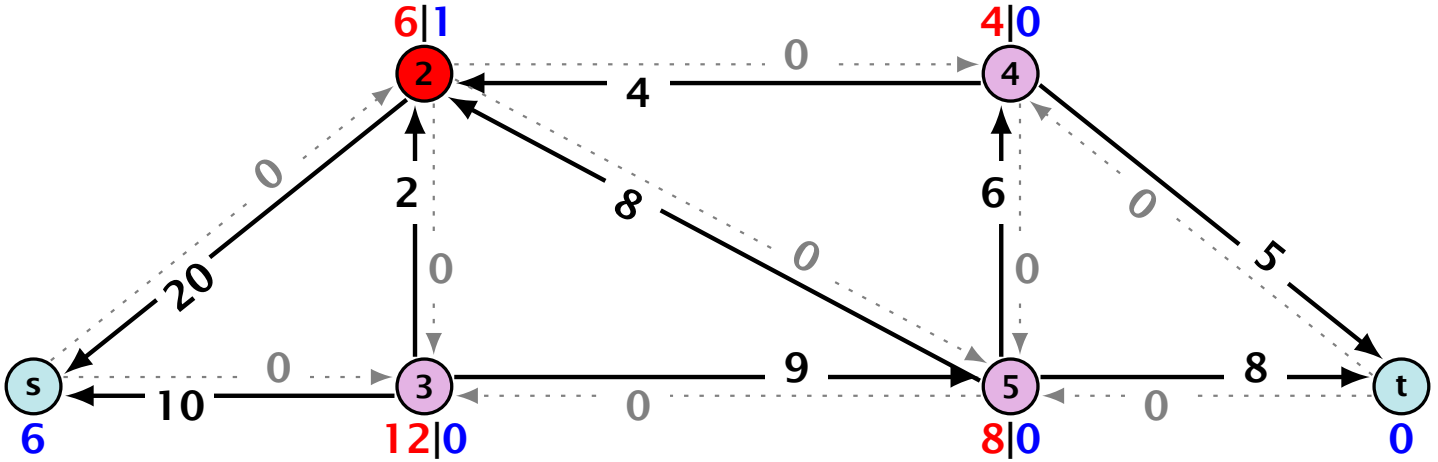
Preflow Push Algorithm

relabel 6 times

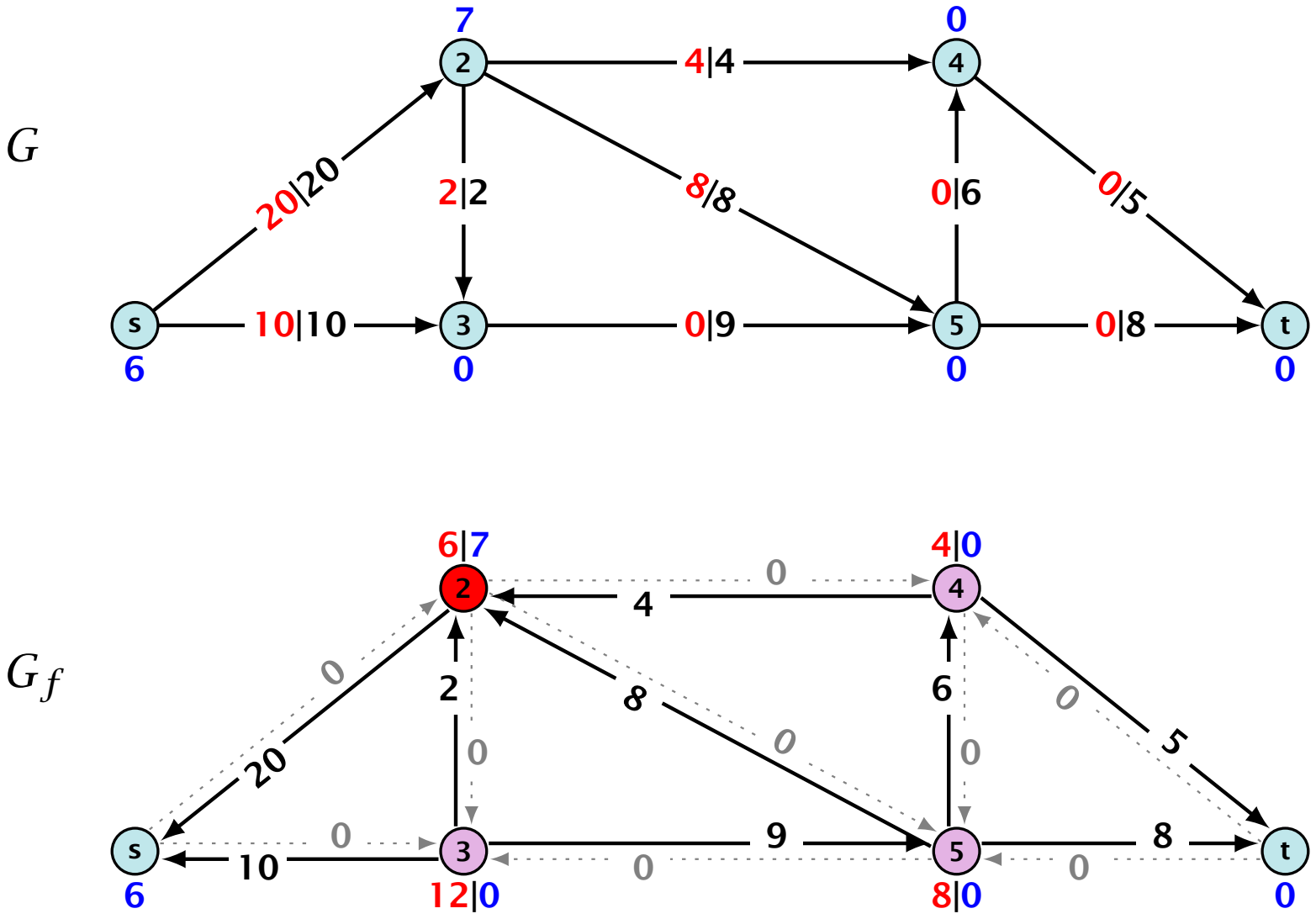
G



G_f

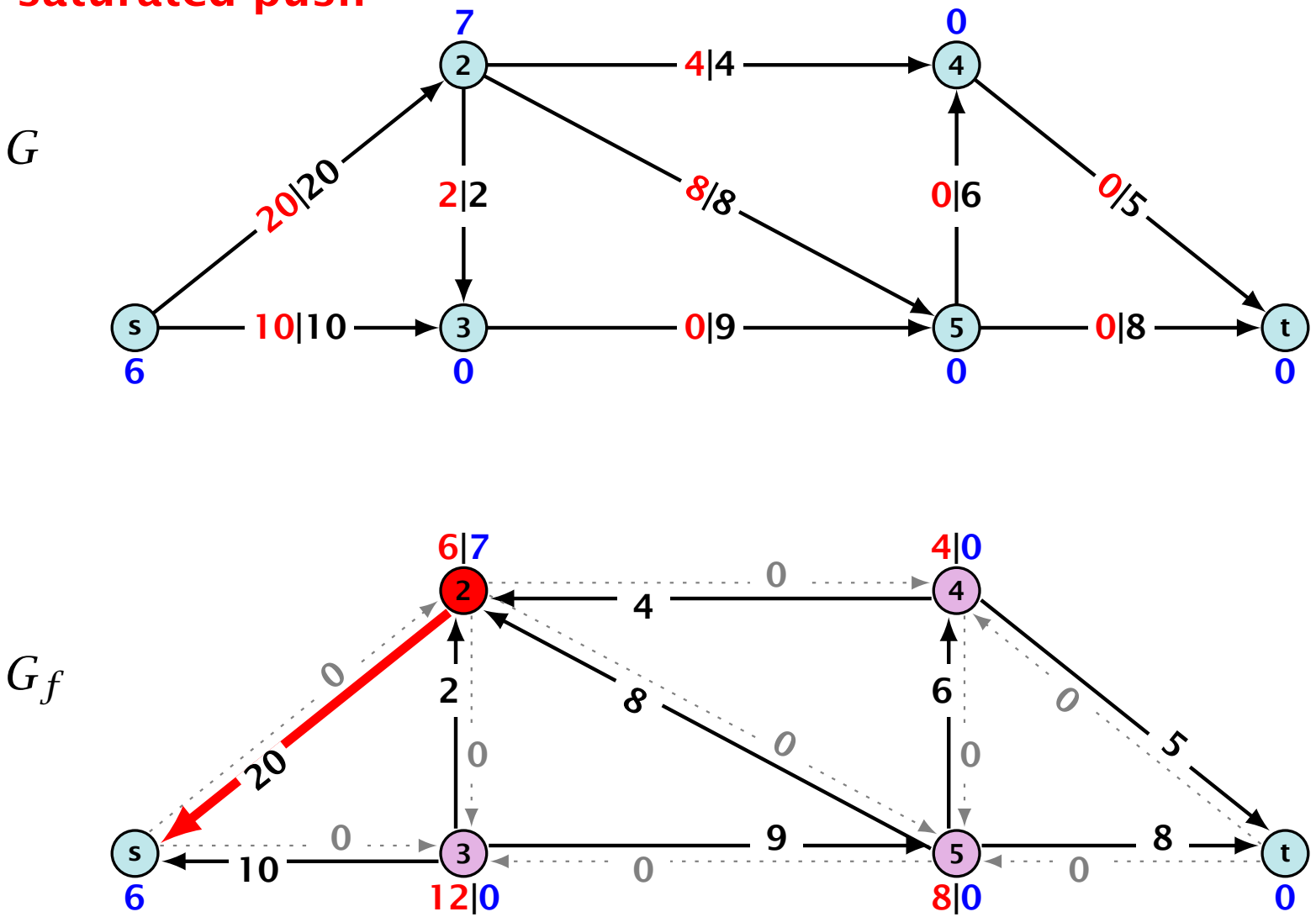


Preflow Push Algorithm

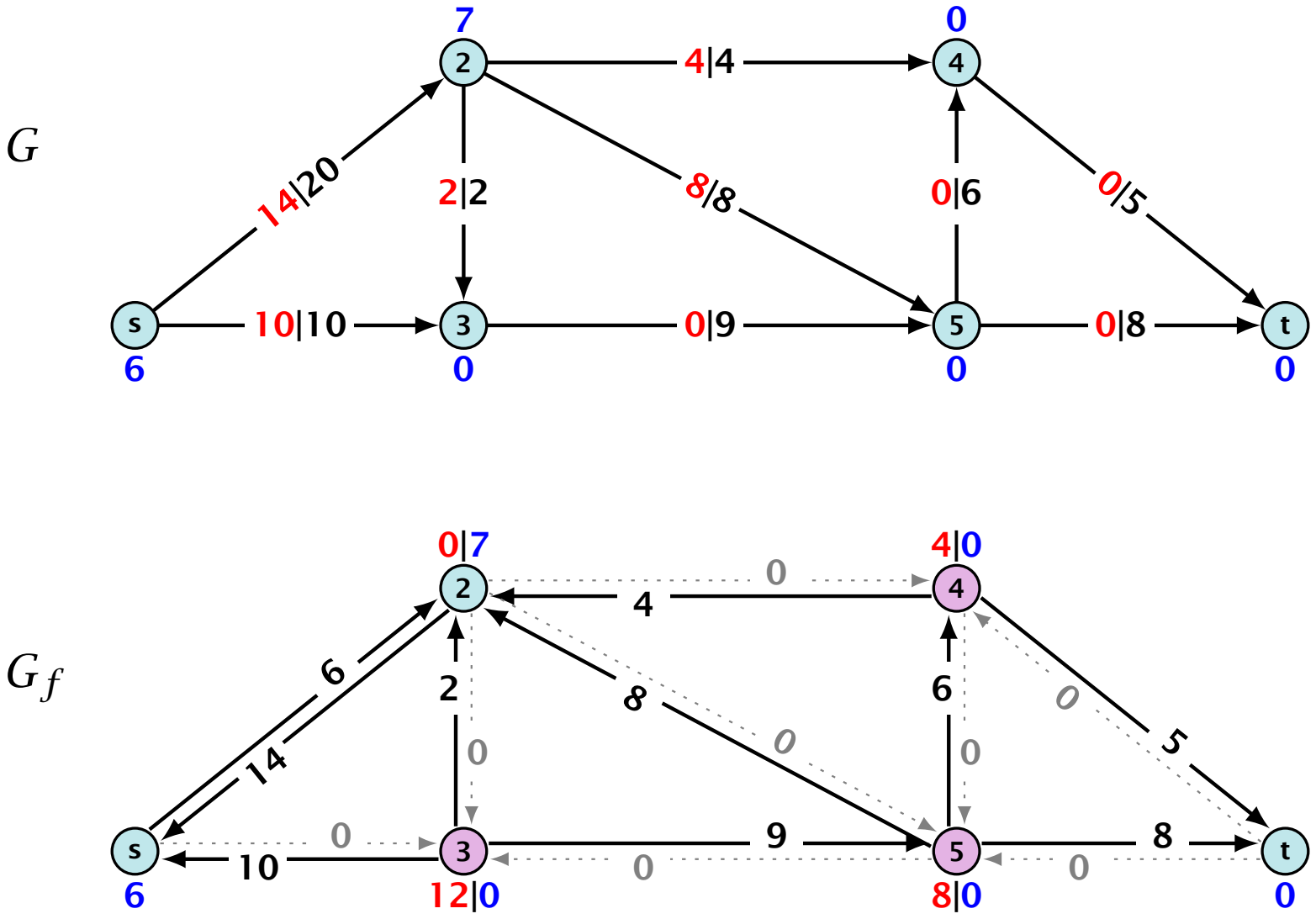


Preflow Push Algorithm

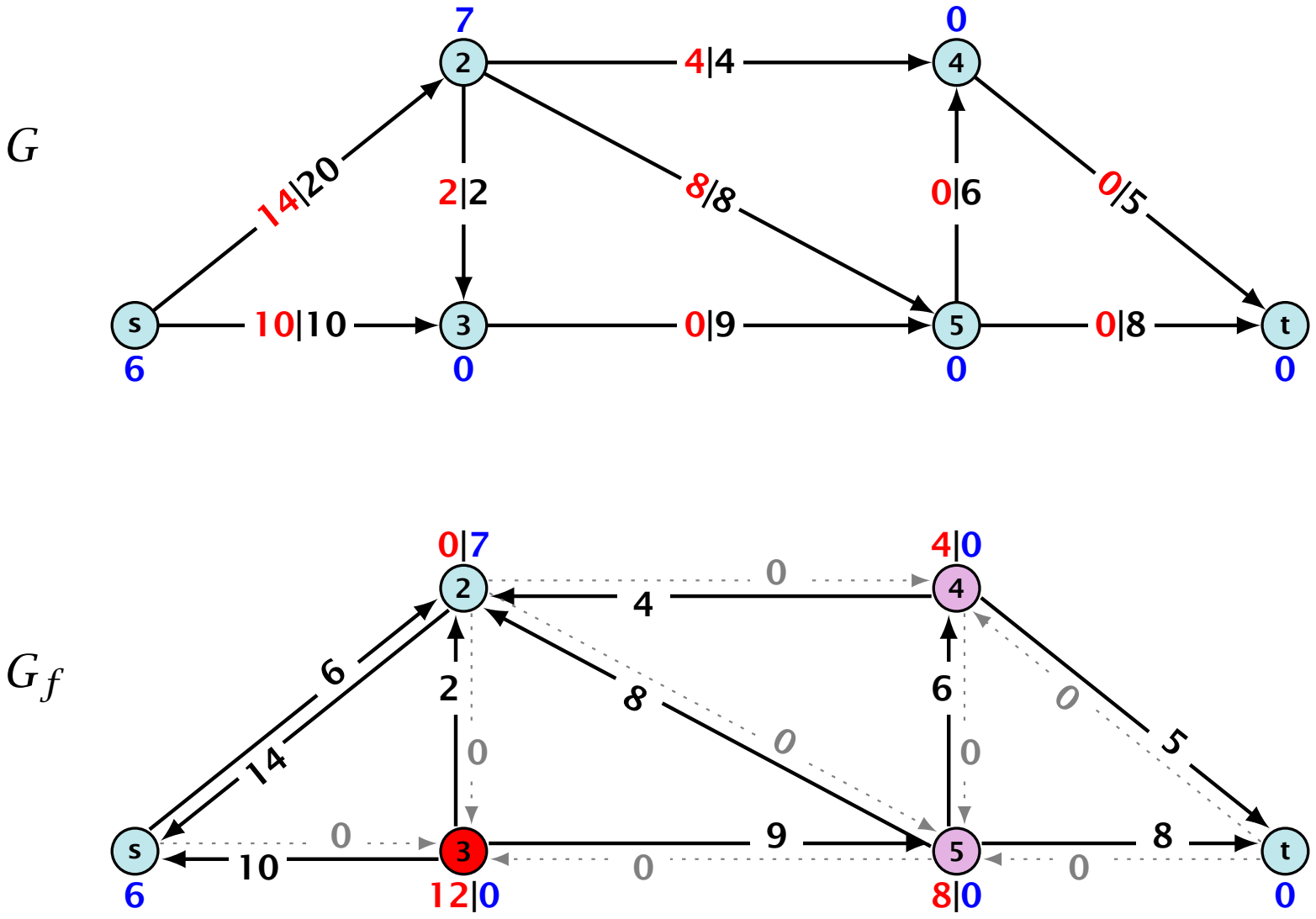
non-saturated push



Preflow Push Algorithm

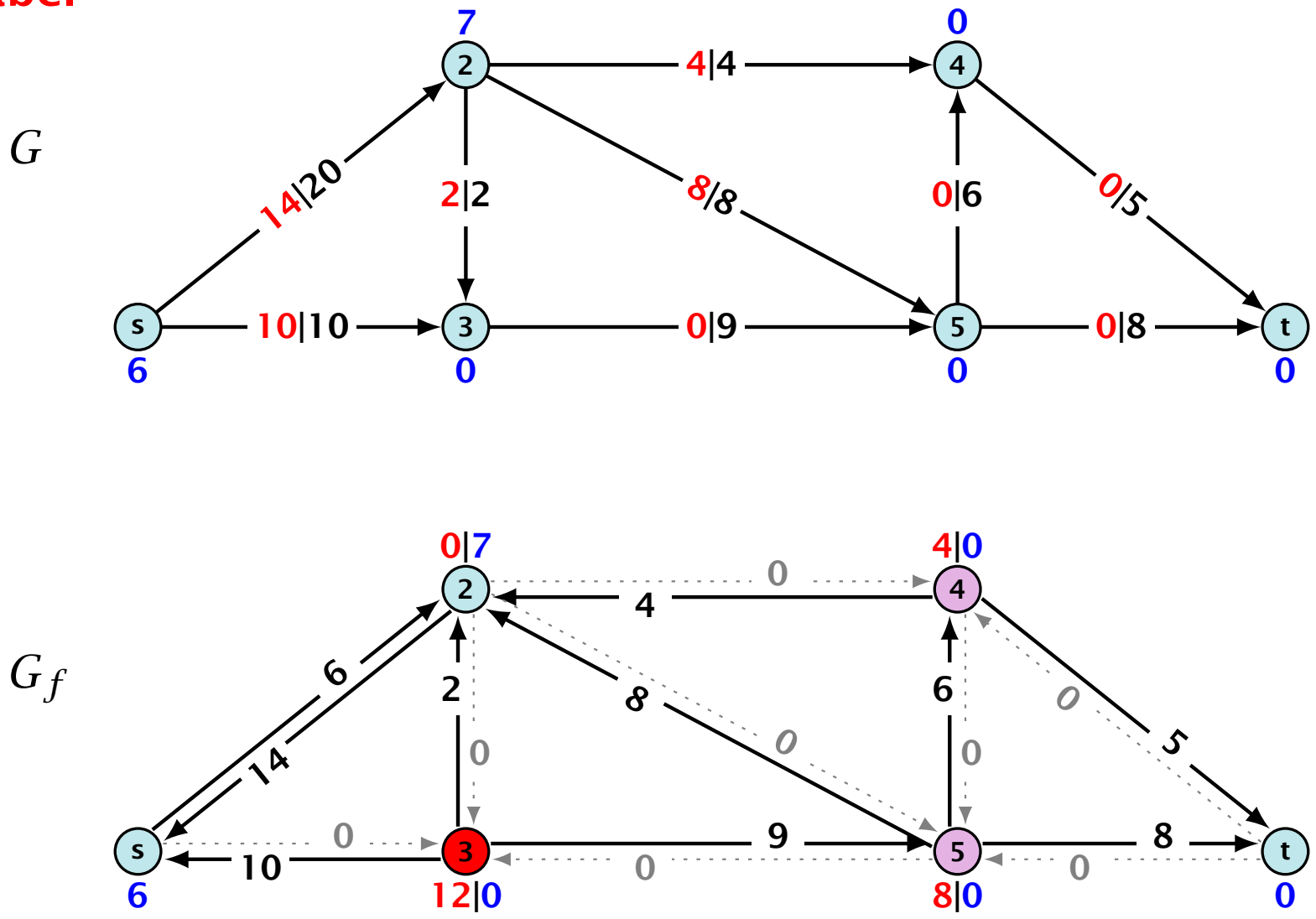


Preflow Push Algorithm

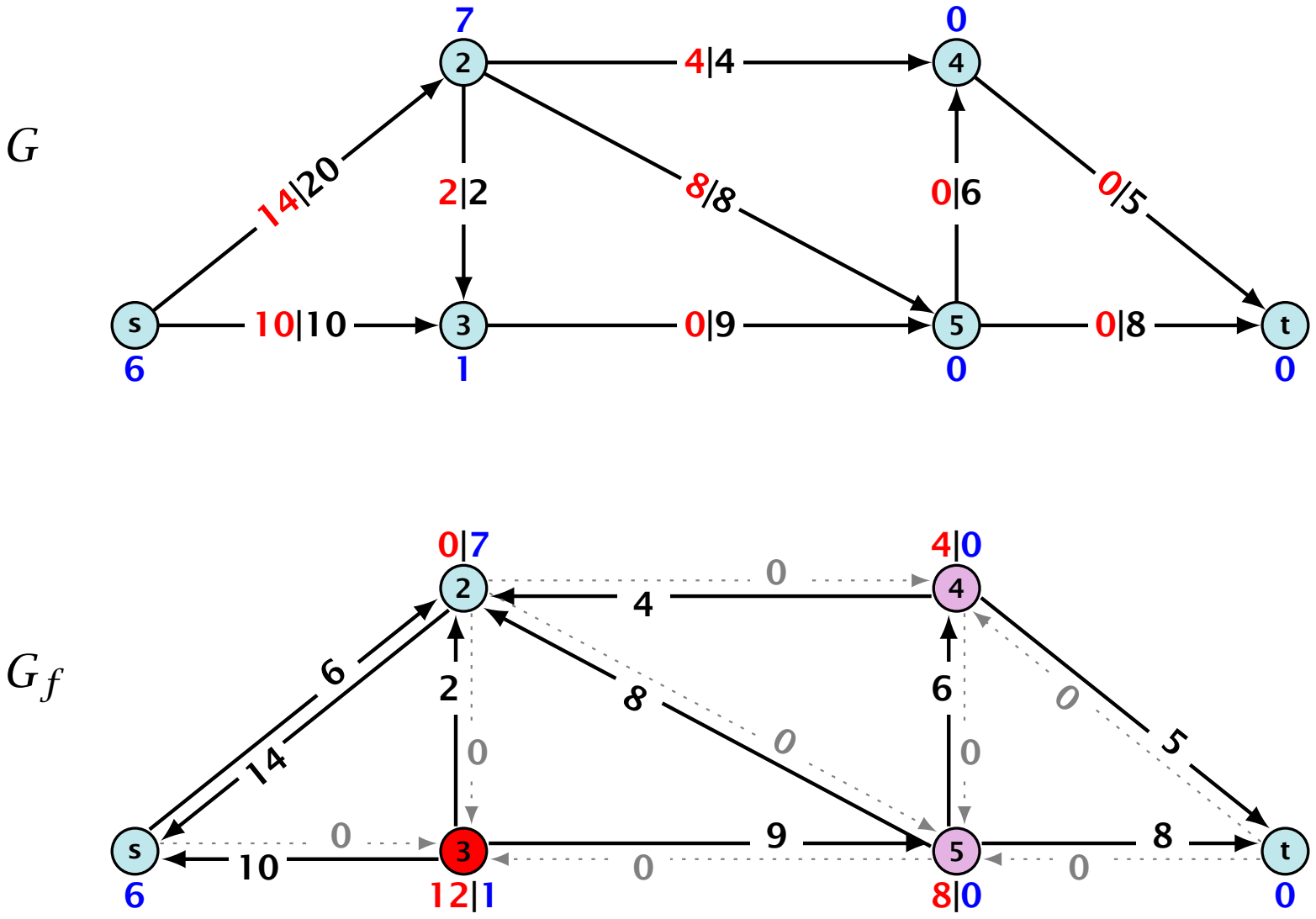


Preflow Push Algorithm

relabel

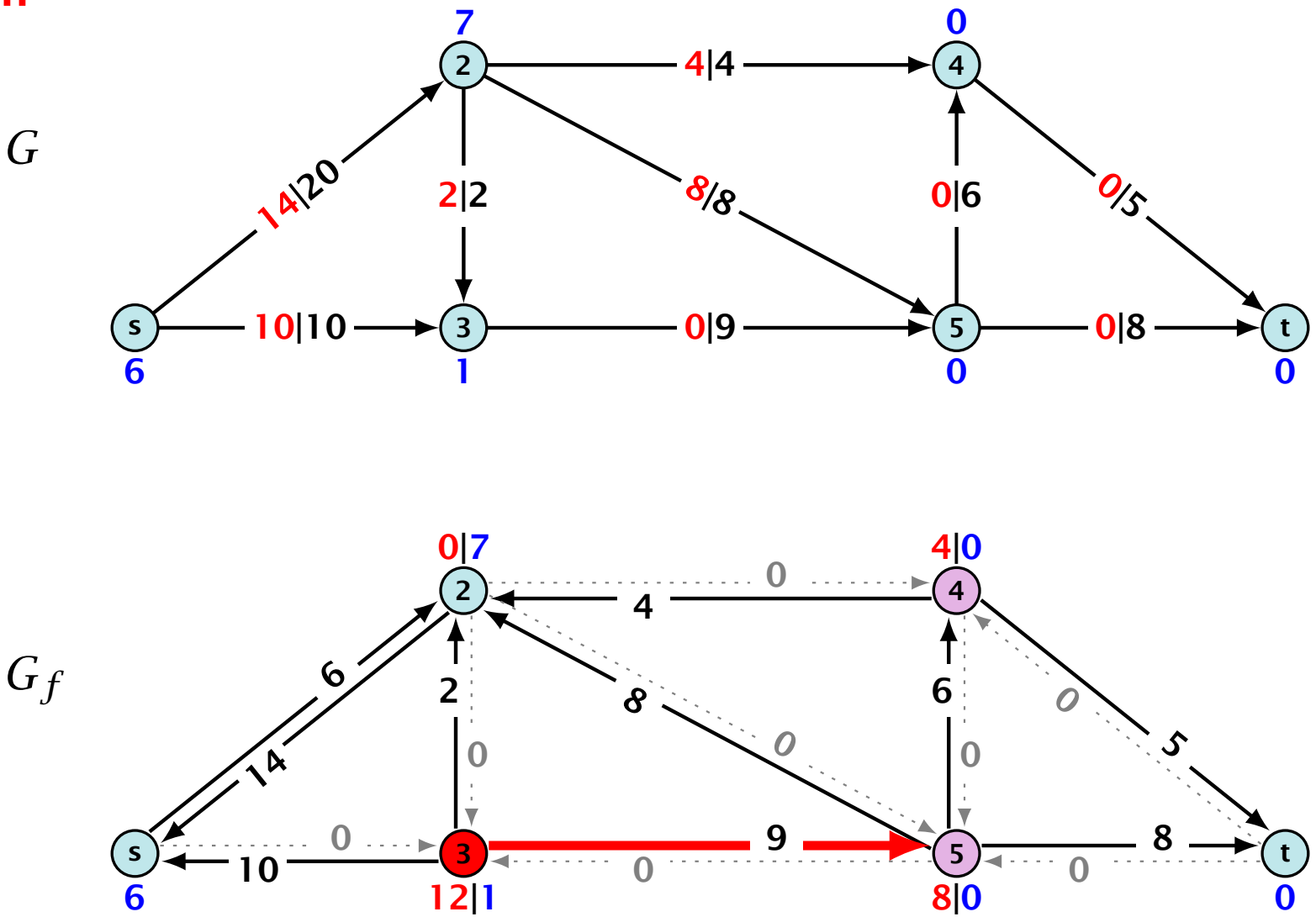


Preflow Push Algorithm

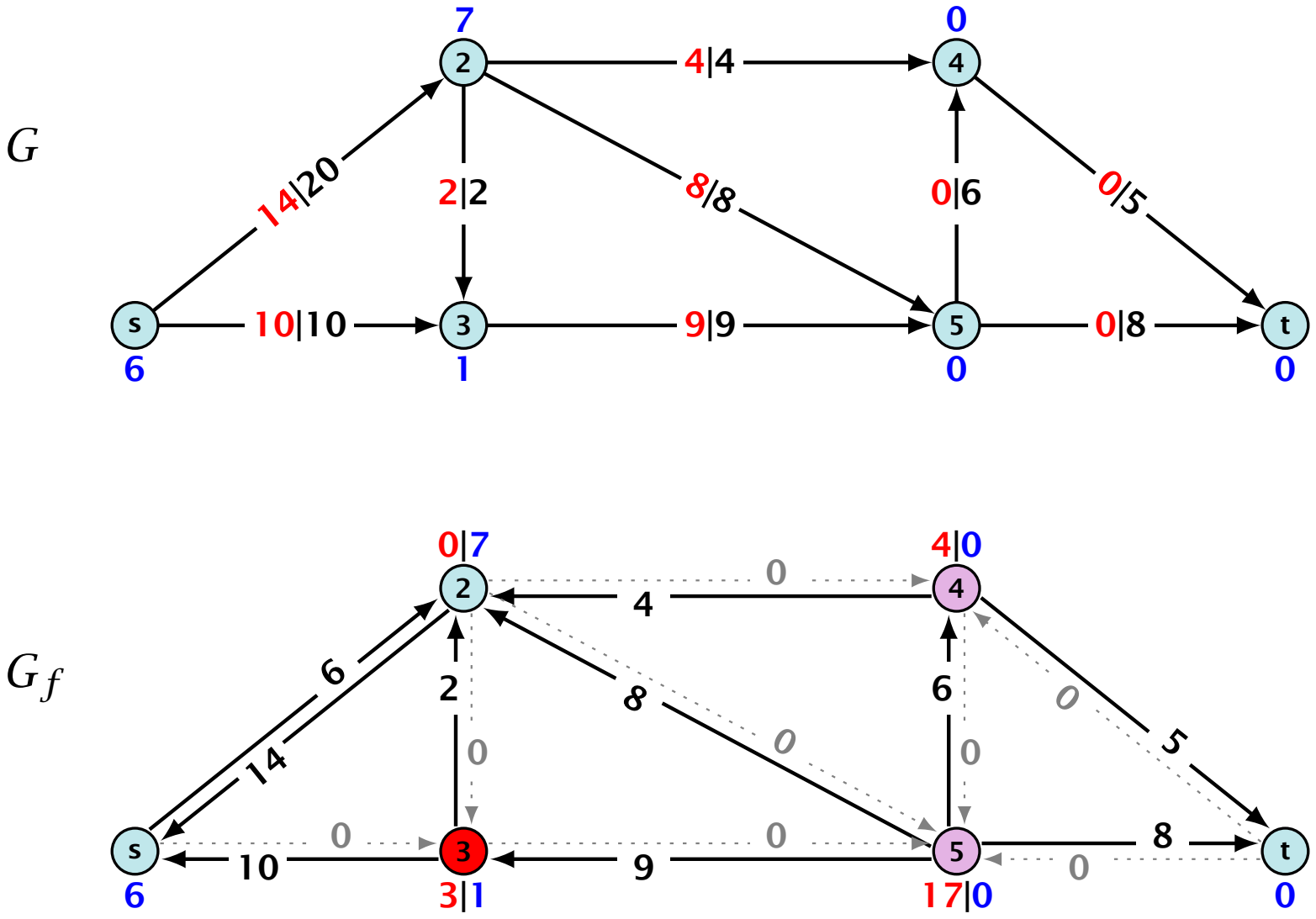


Preflow Push Algorithm

push

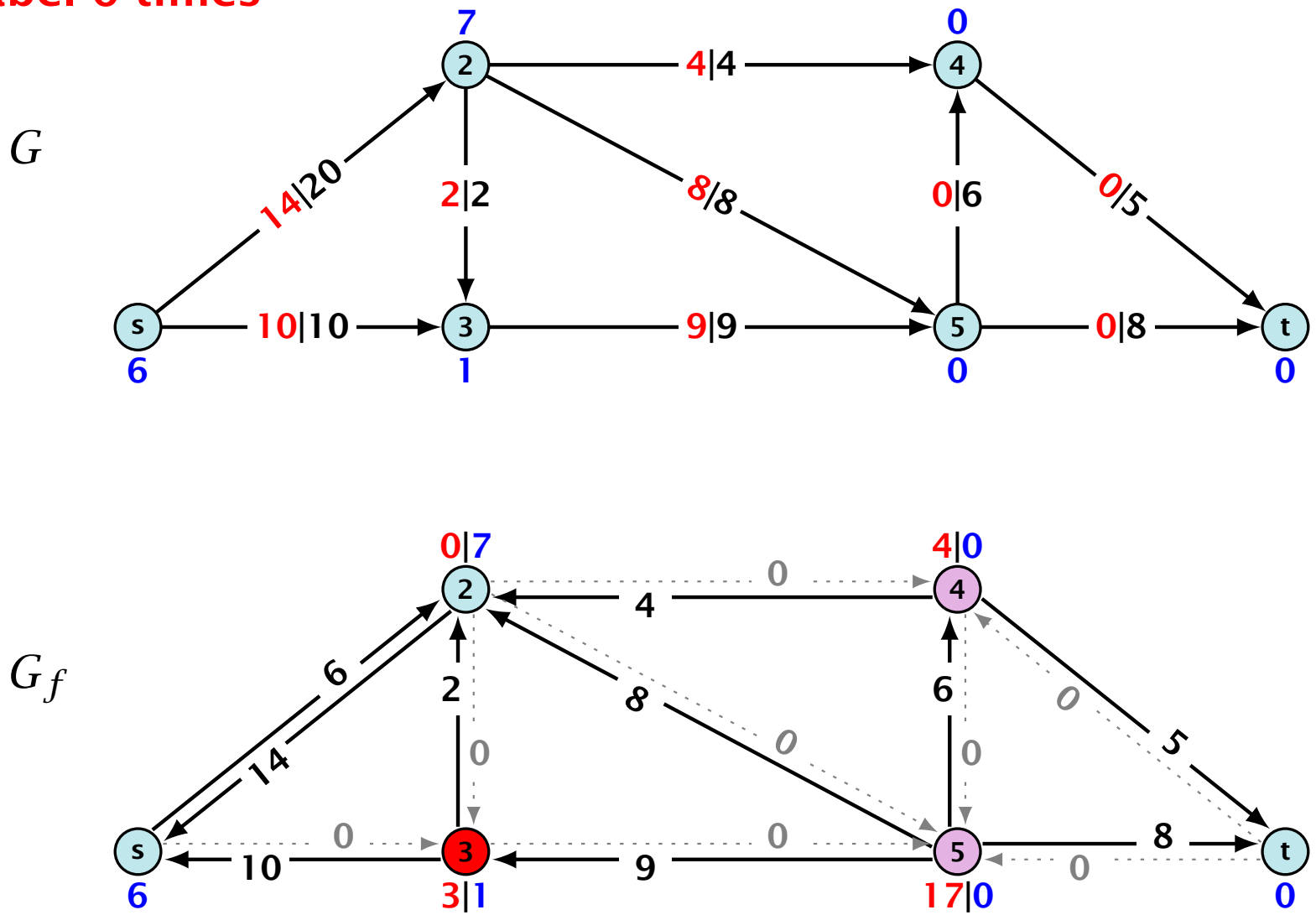


Preflow Push Algorithm

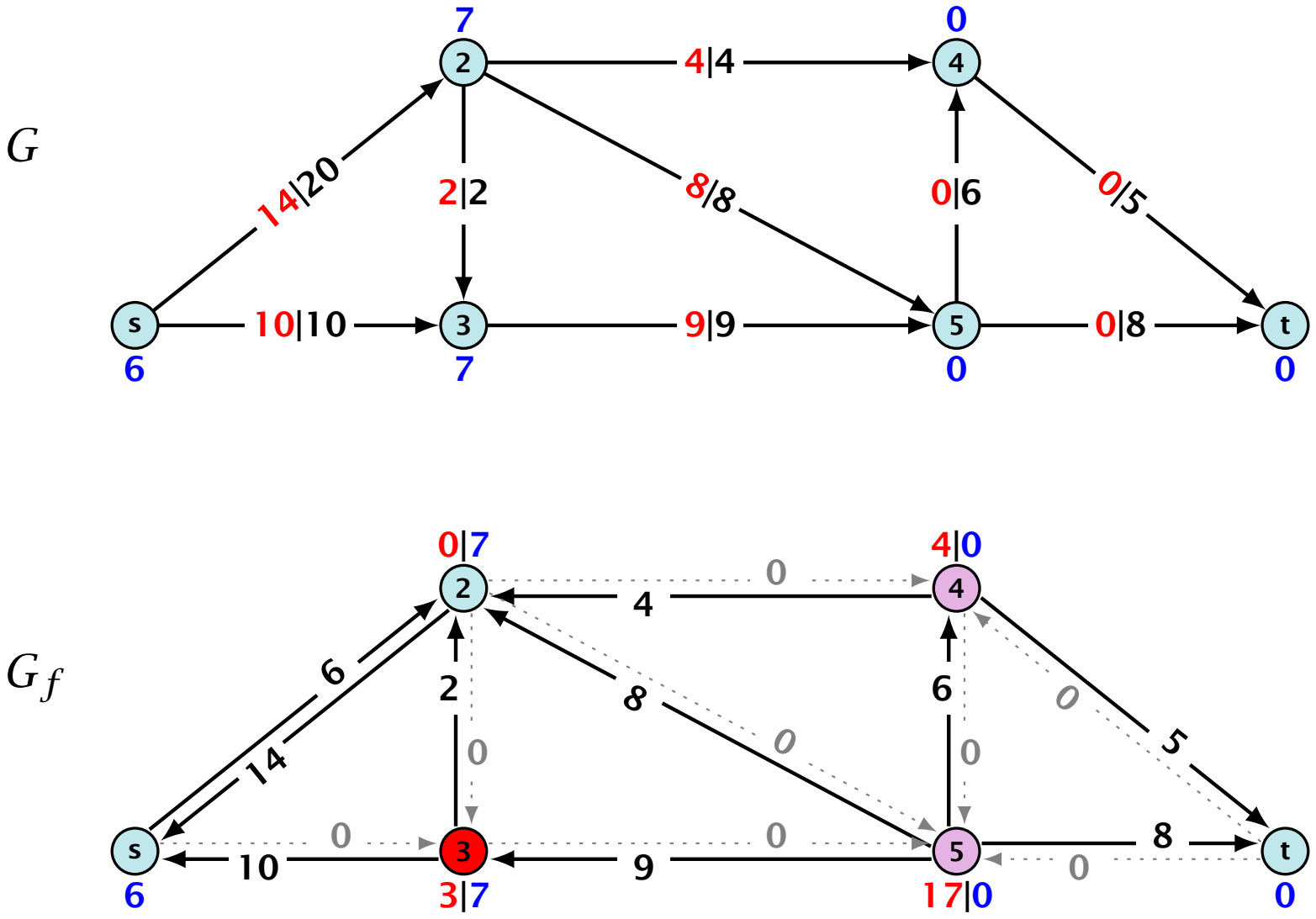


Preflow Push Algorithm

relabel 6 times

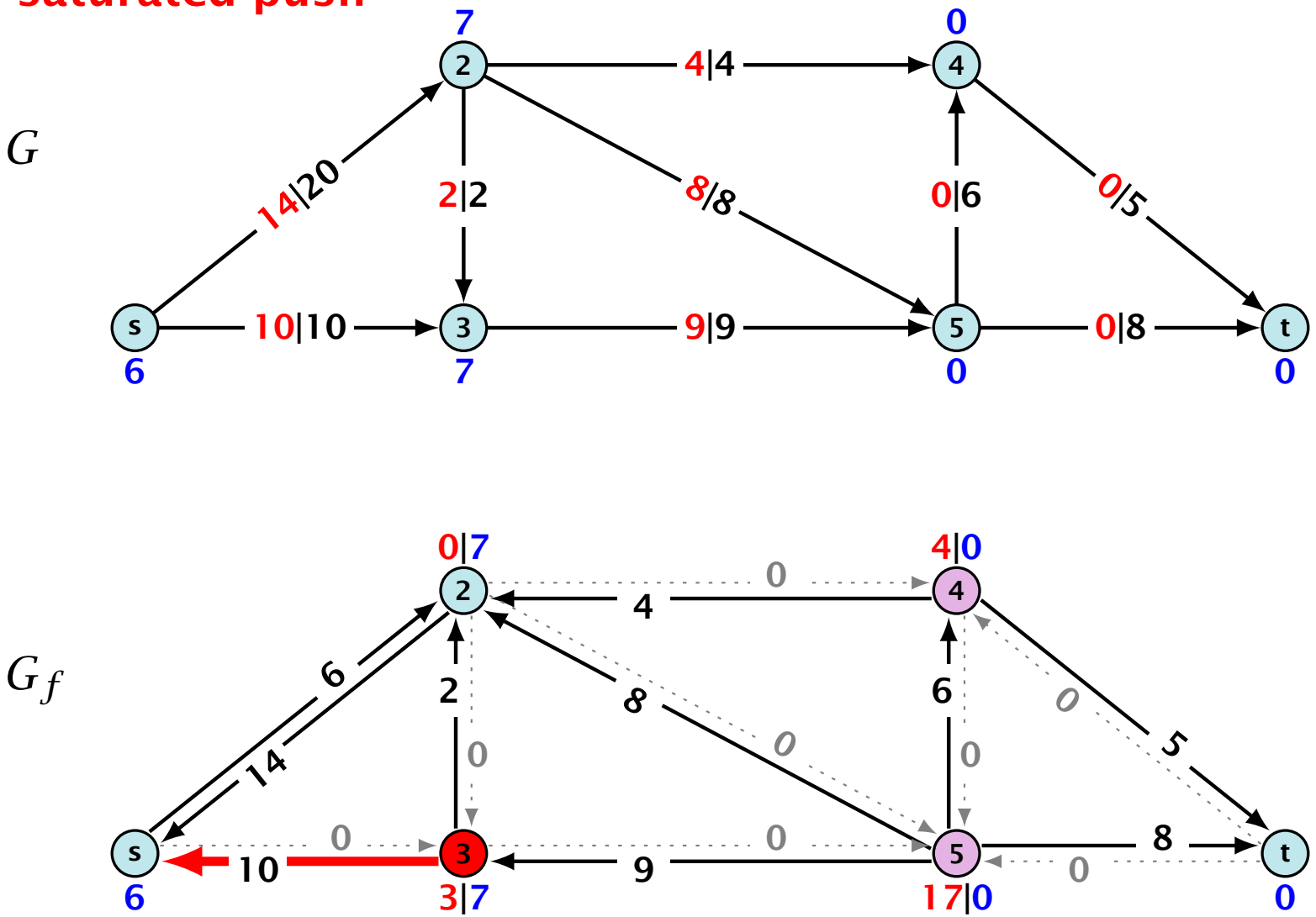


Preflow Push Algorithm

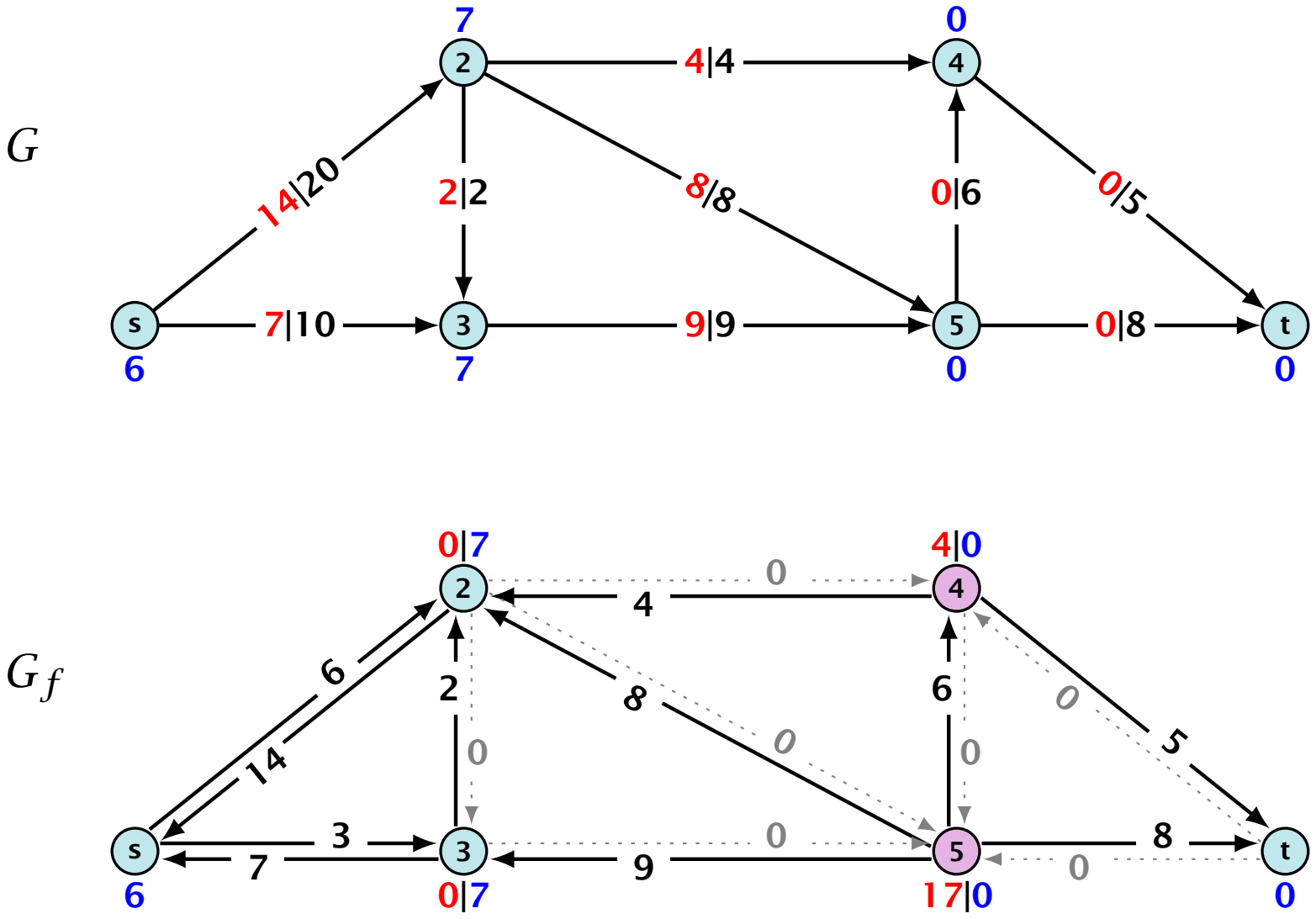


Preflow Push Algorithm

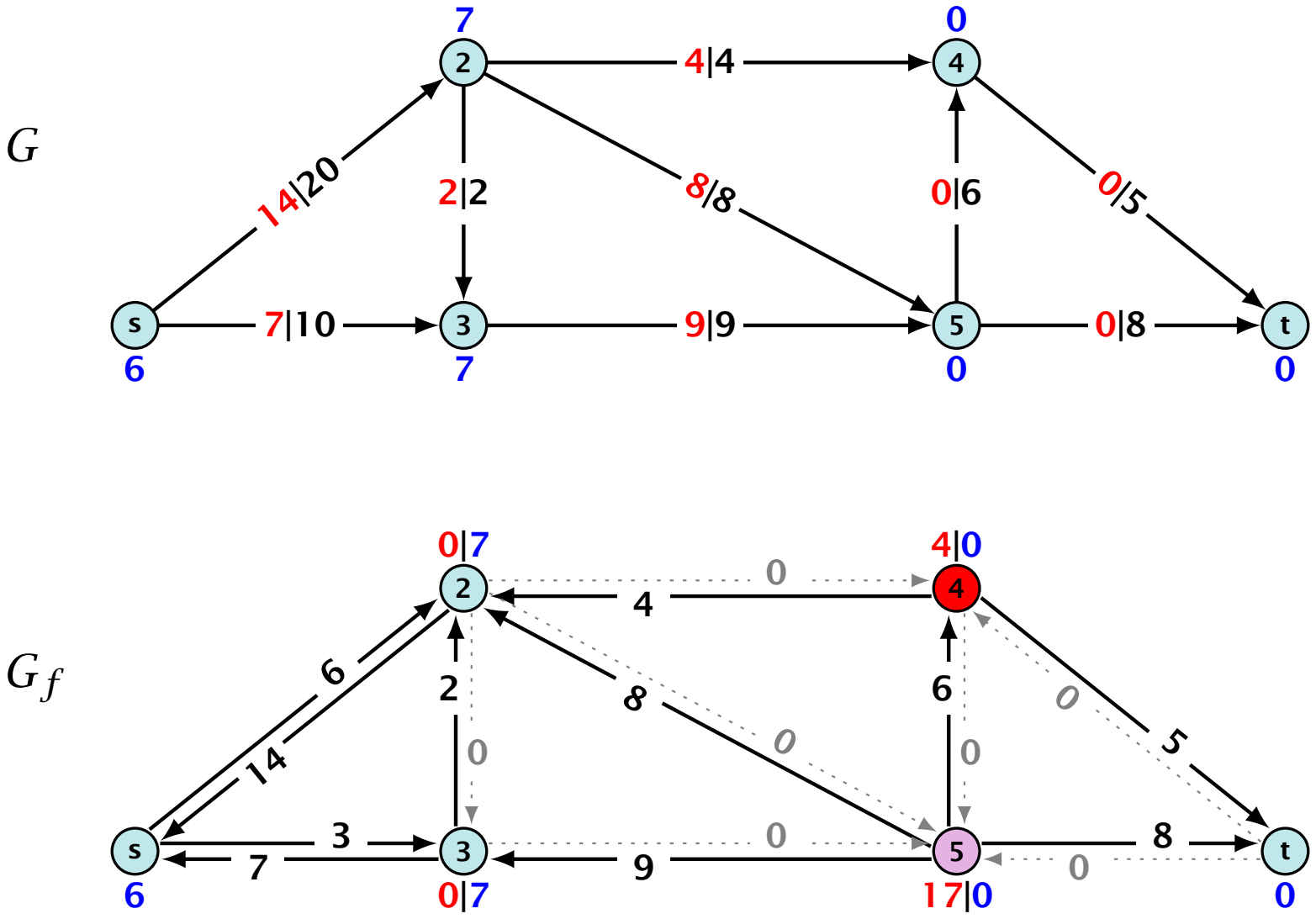
non-saturated push



Preflow Push Algorithm

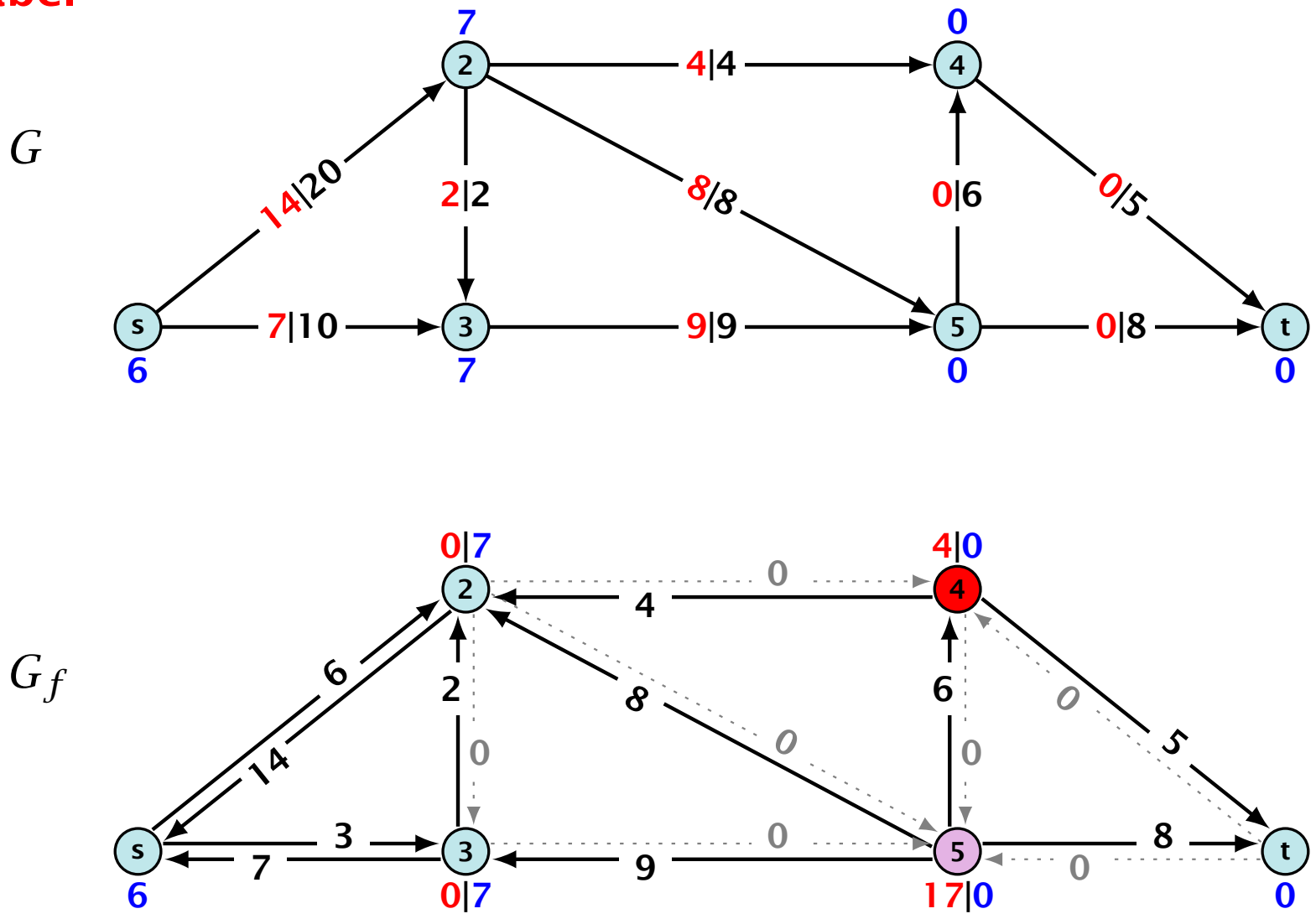


Preflow Push Algorithm

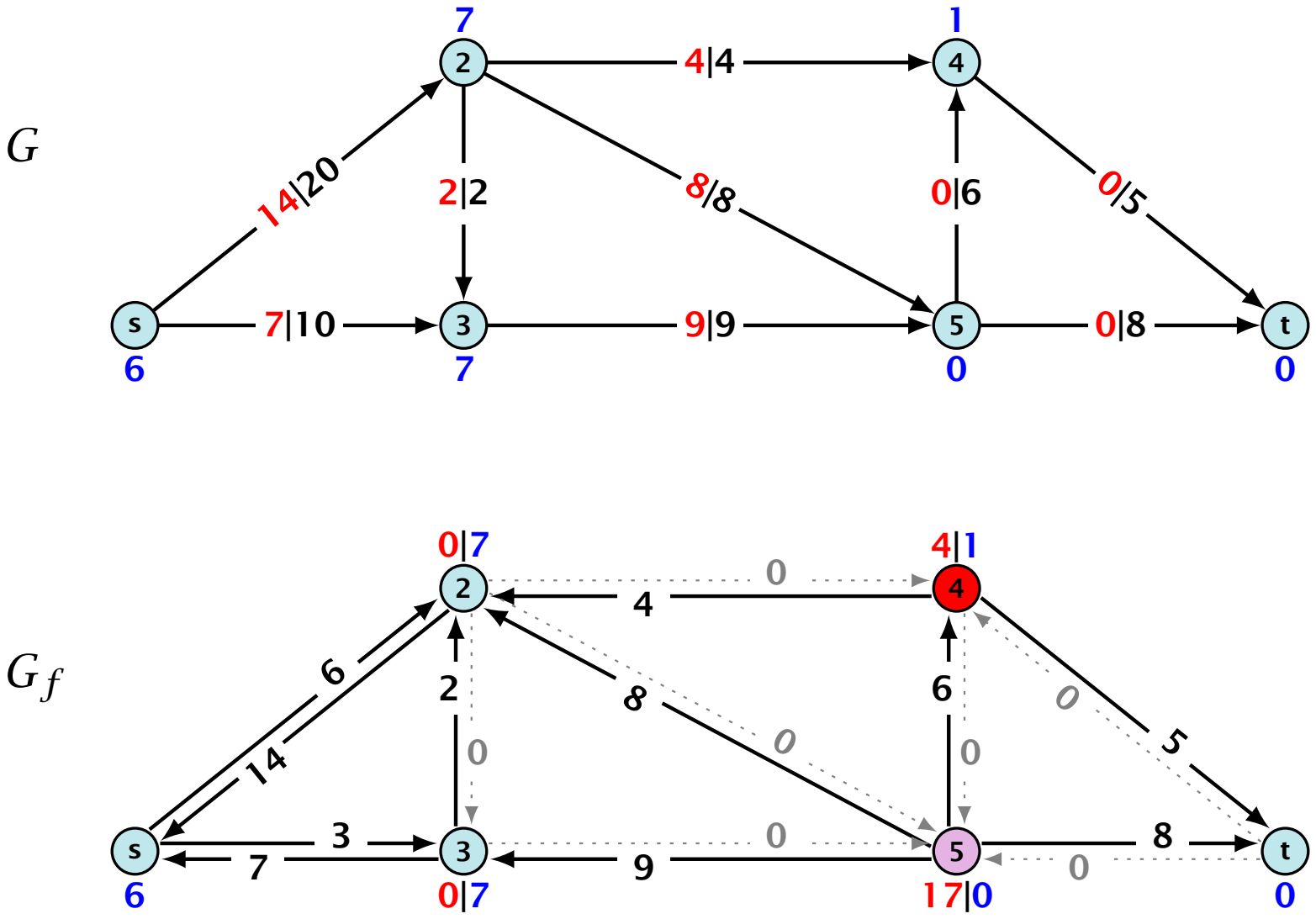


Preflow Push Algorithm

relabel

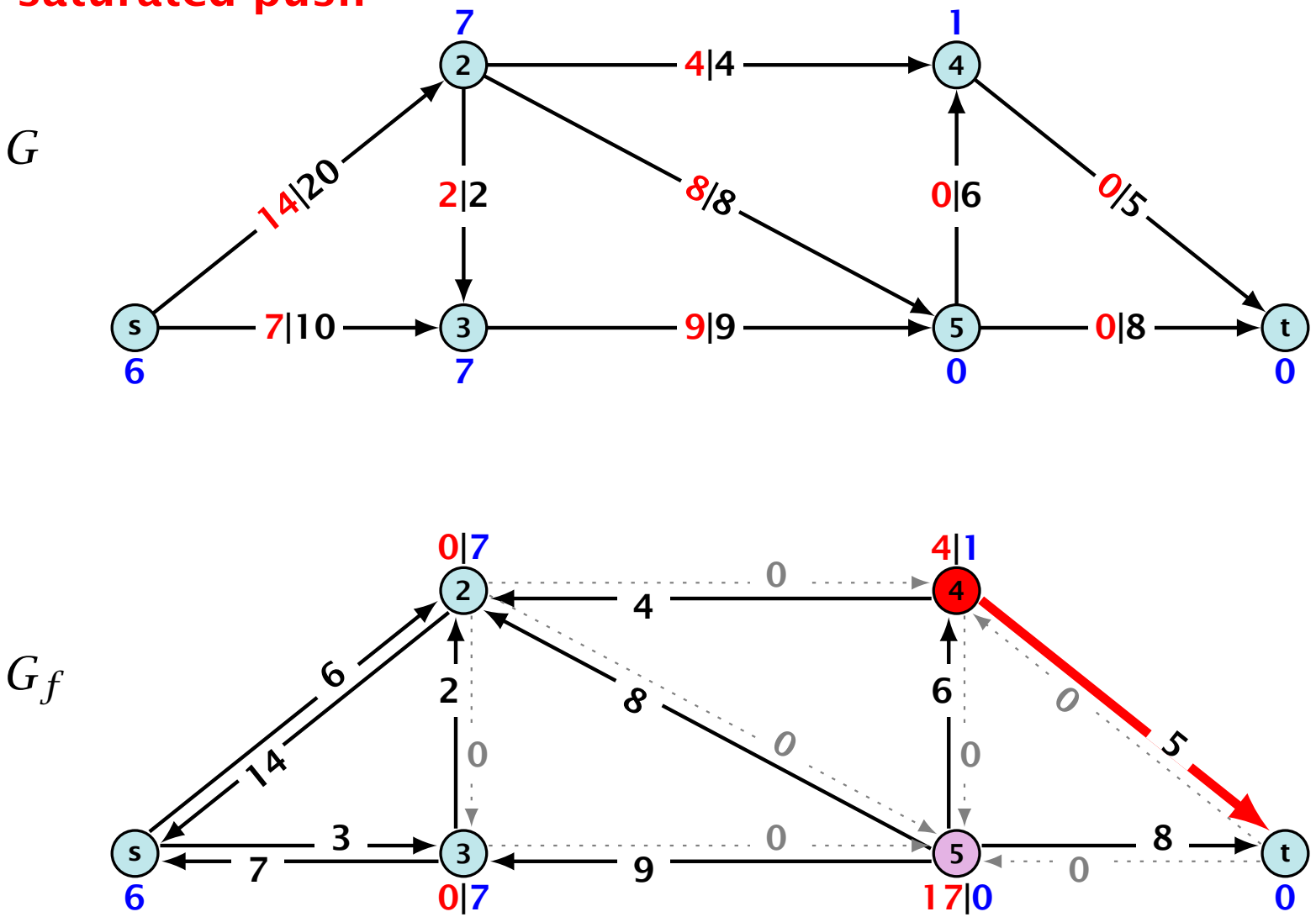


Preflow Push Algorithm

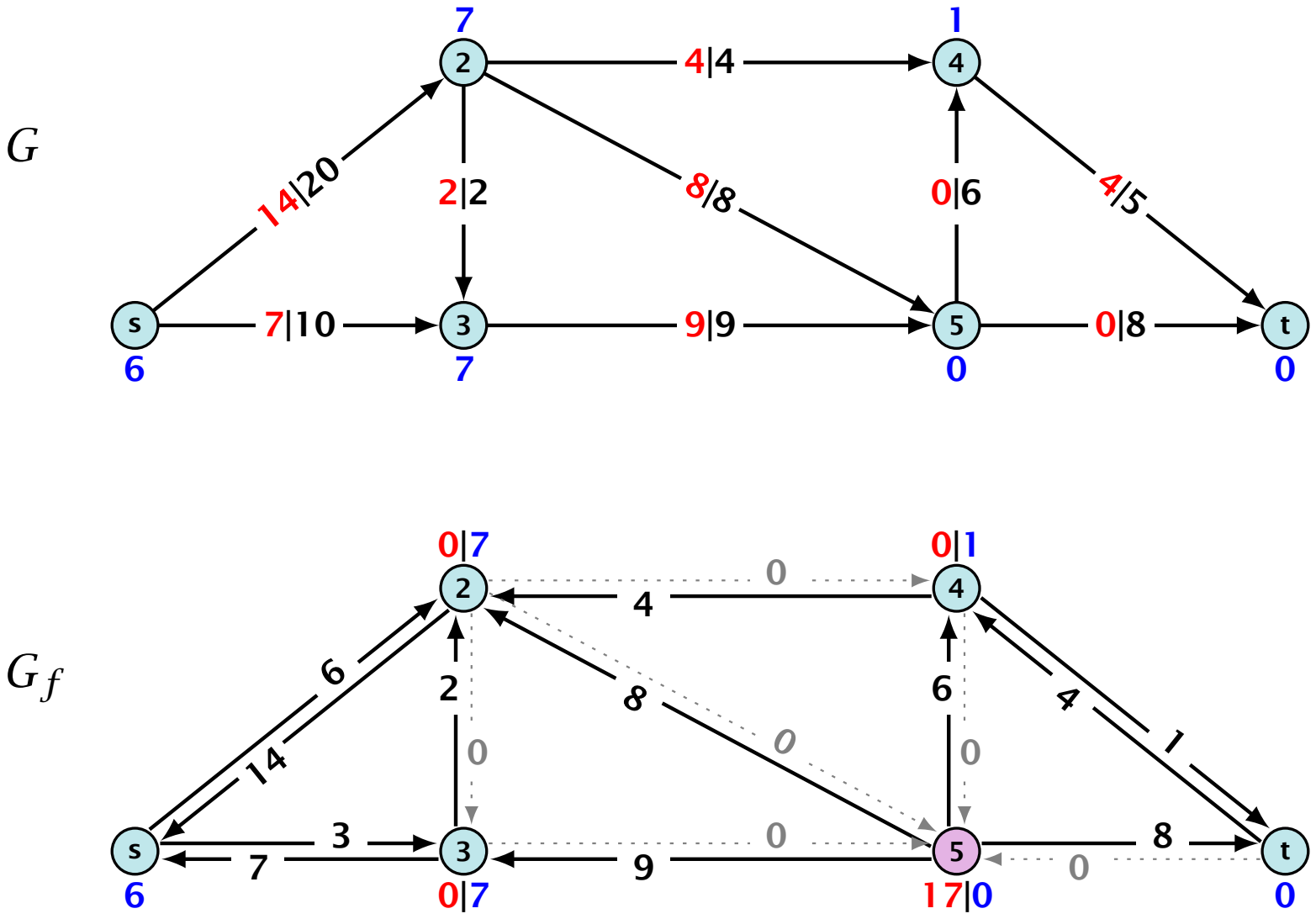


Preflow Push Algorithm

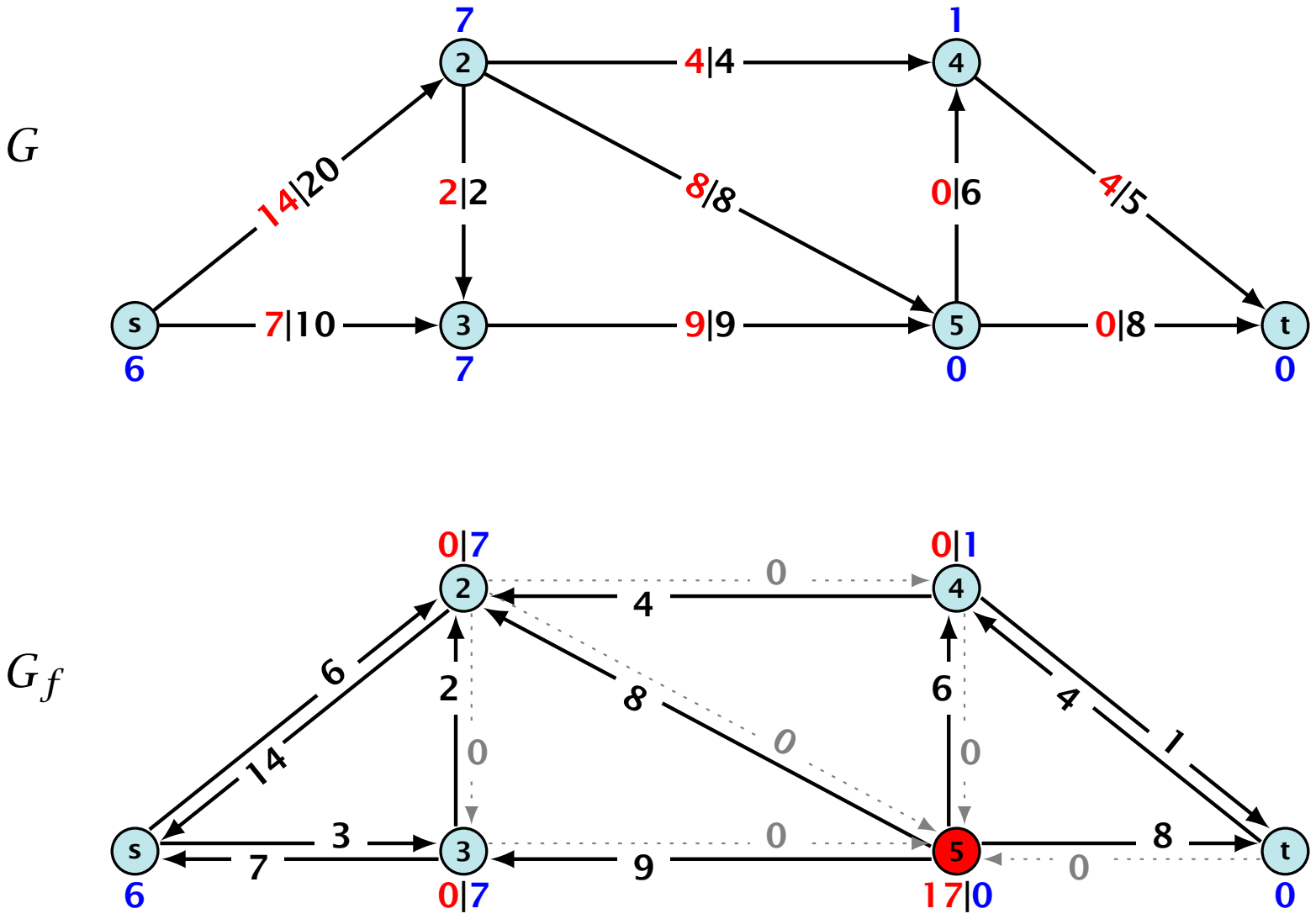
non-saturated push



Preflow Push Algorithm

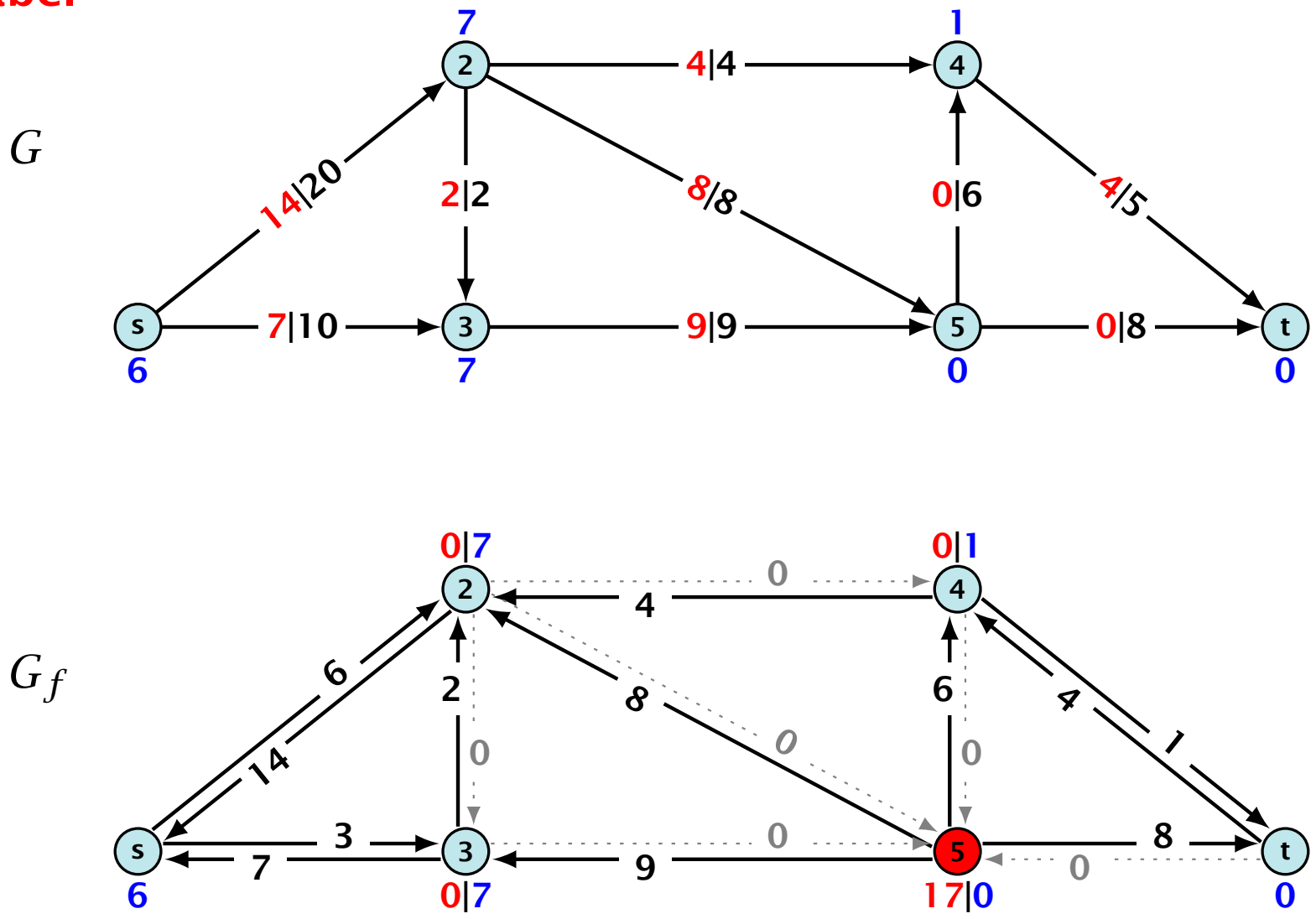


Preflow Push Algorithm

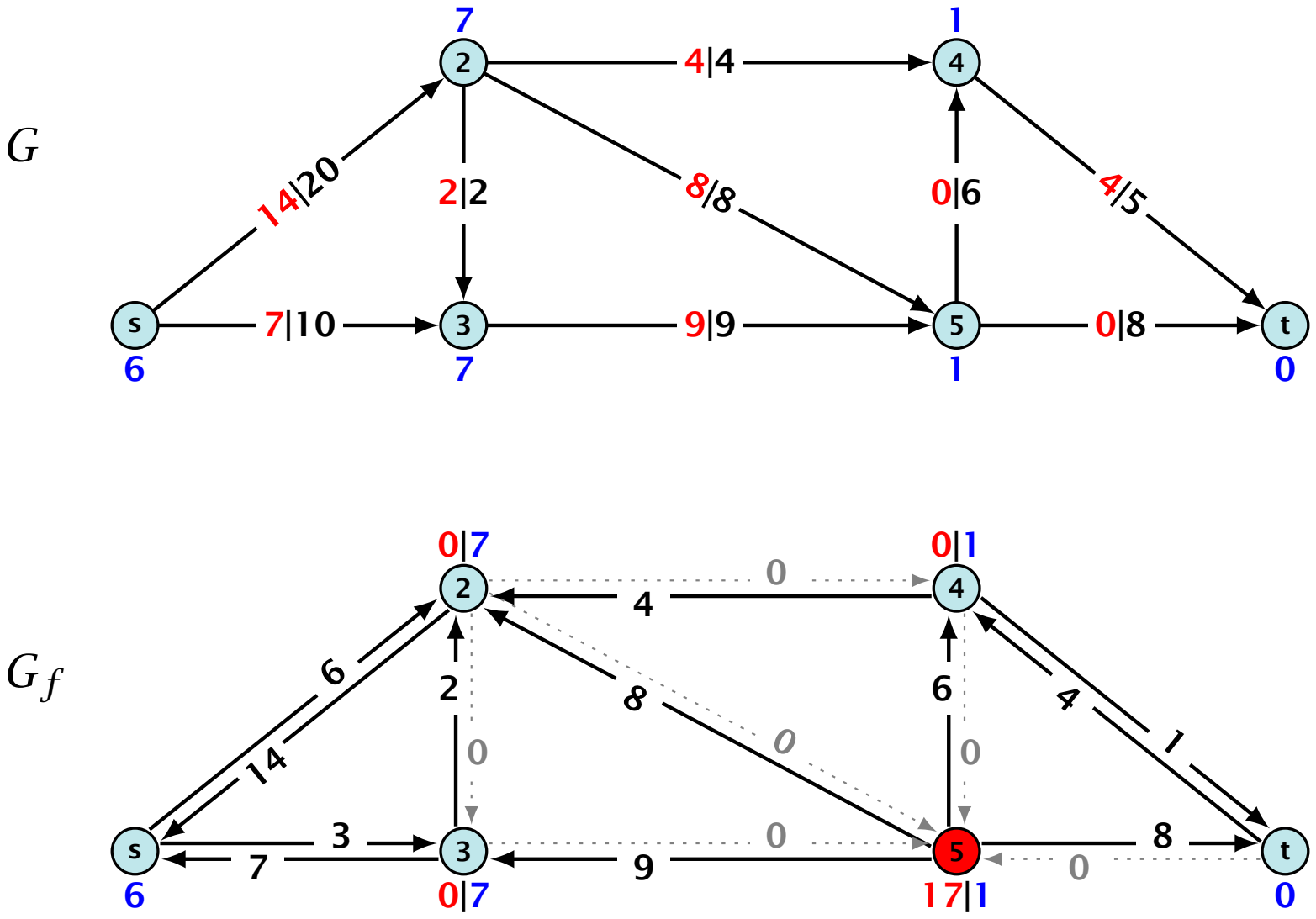


Preflow Push Algorithm

relabel

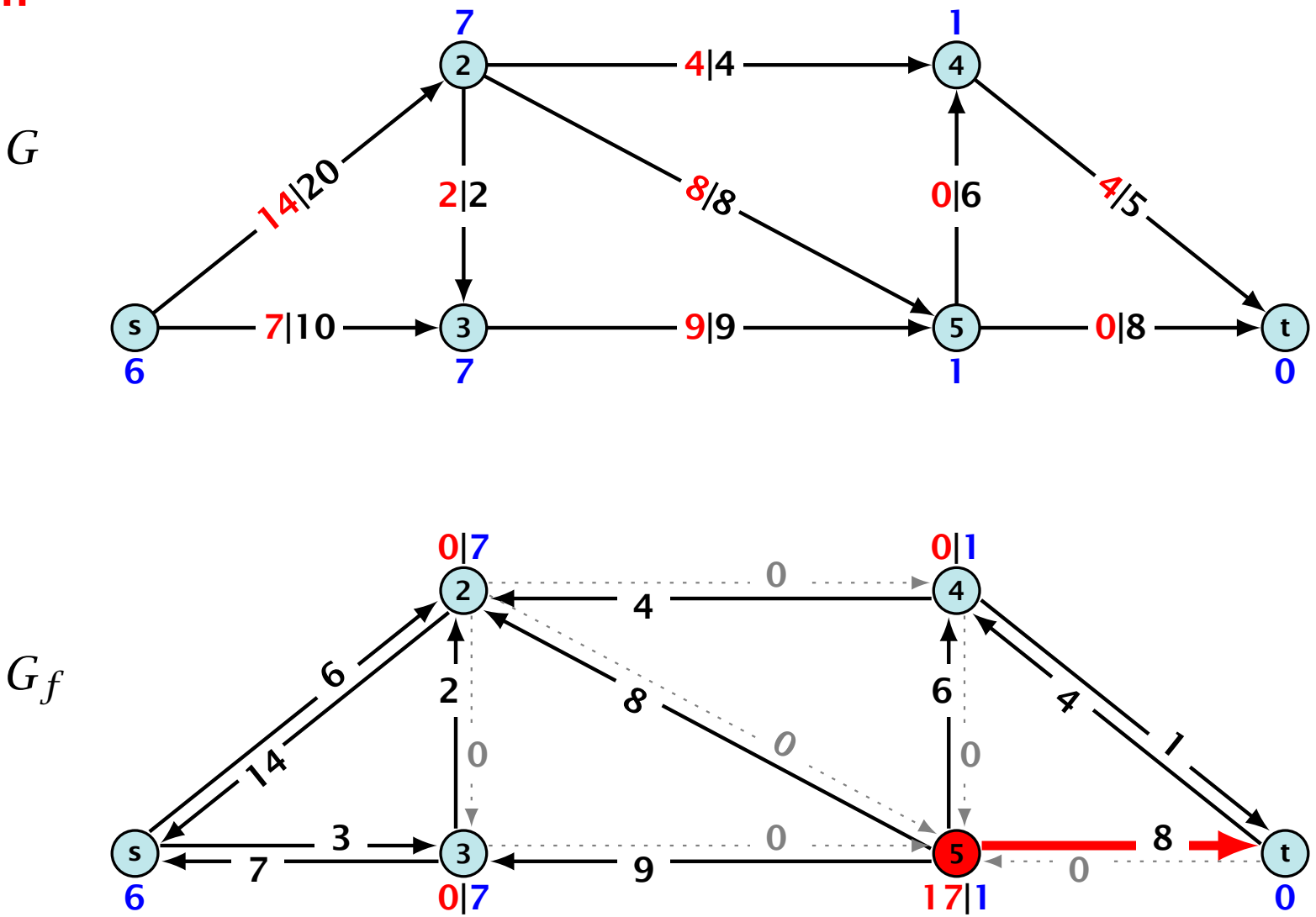


Preflow Push Algorithm

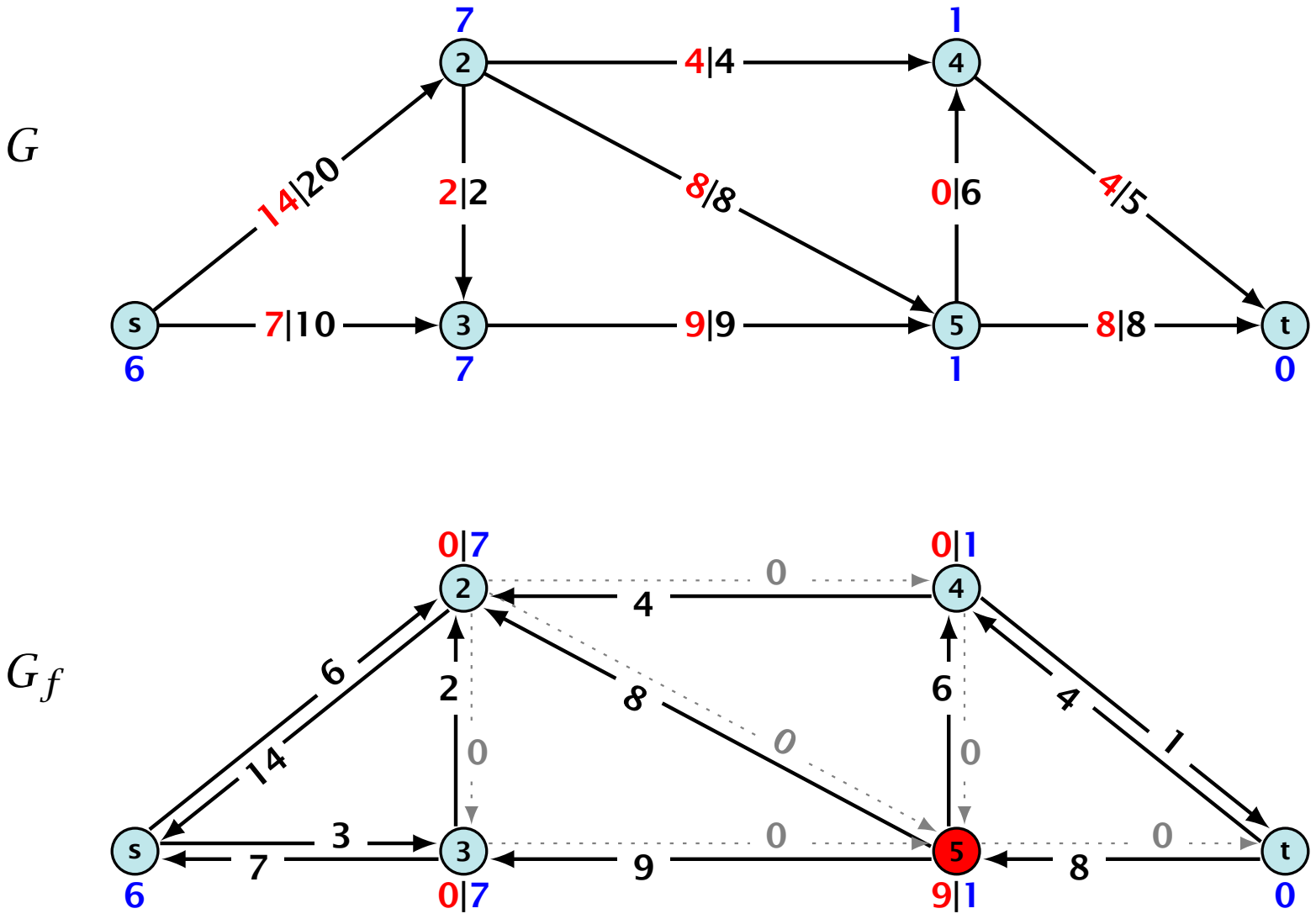


Preflow Push Algorithm

push

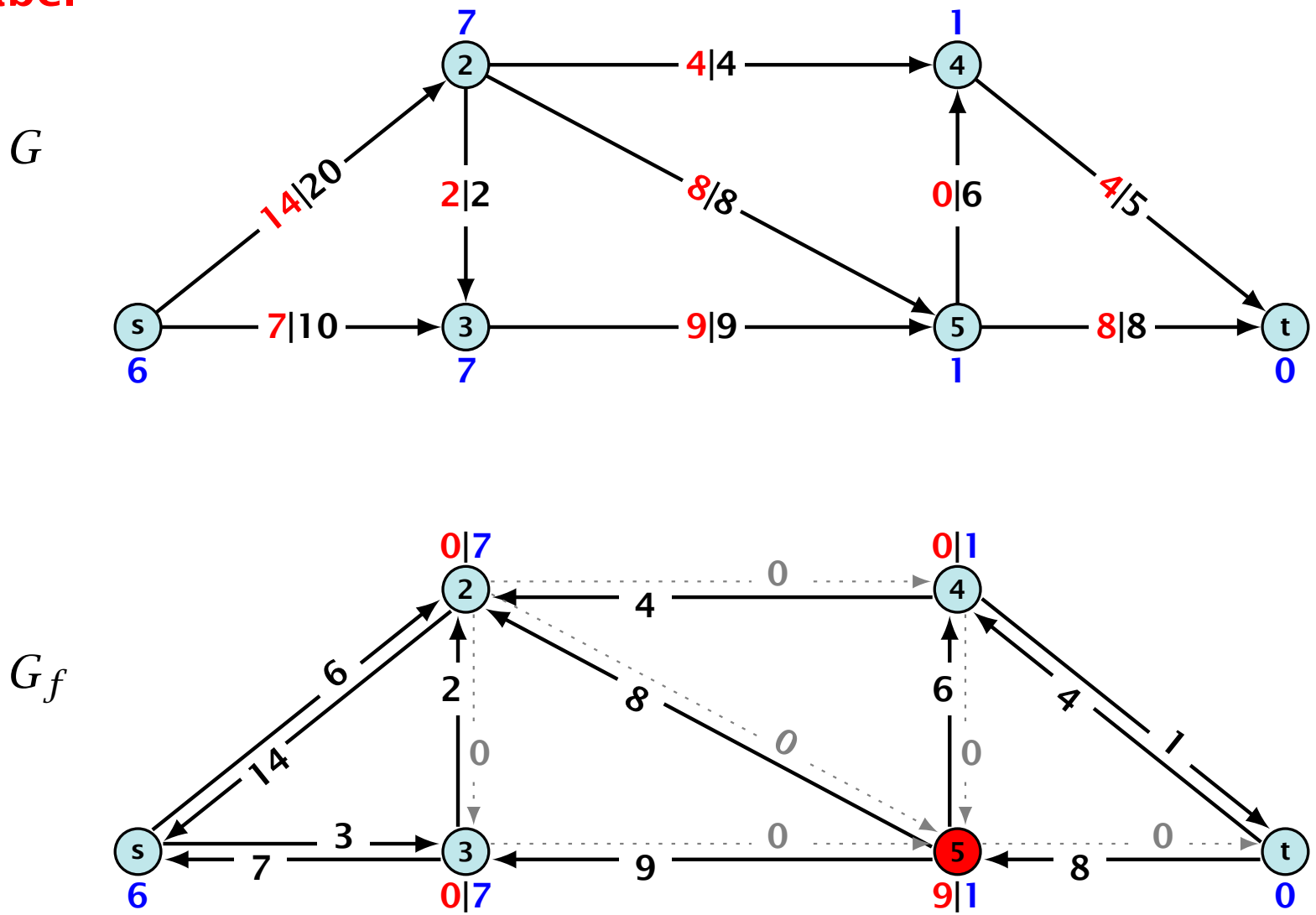


Preflow Push Algorithm

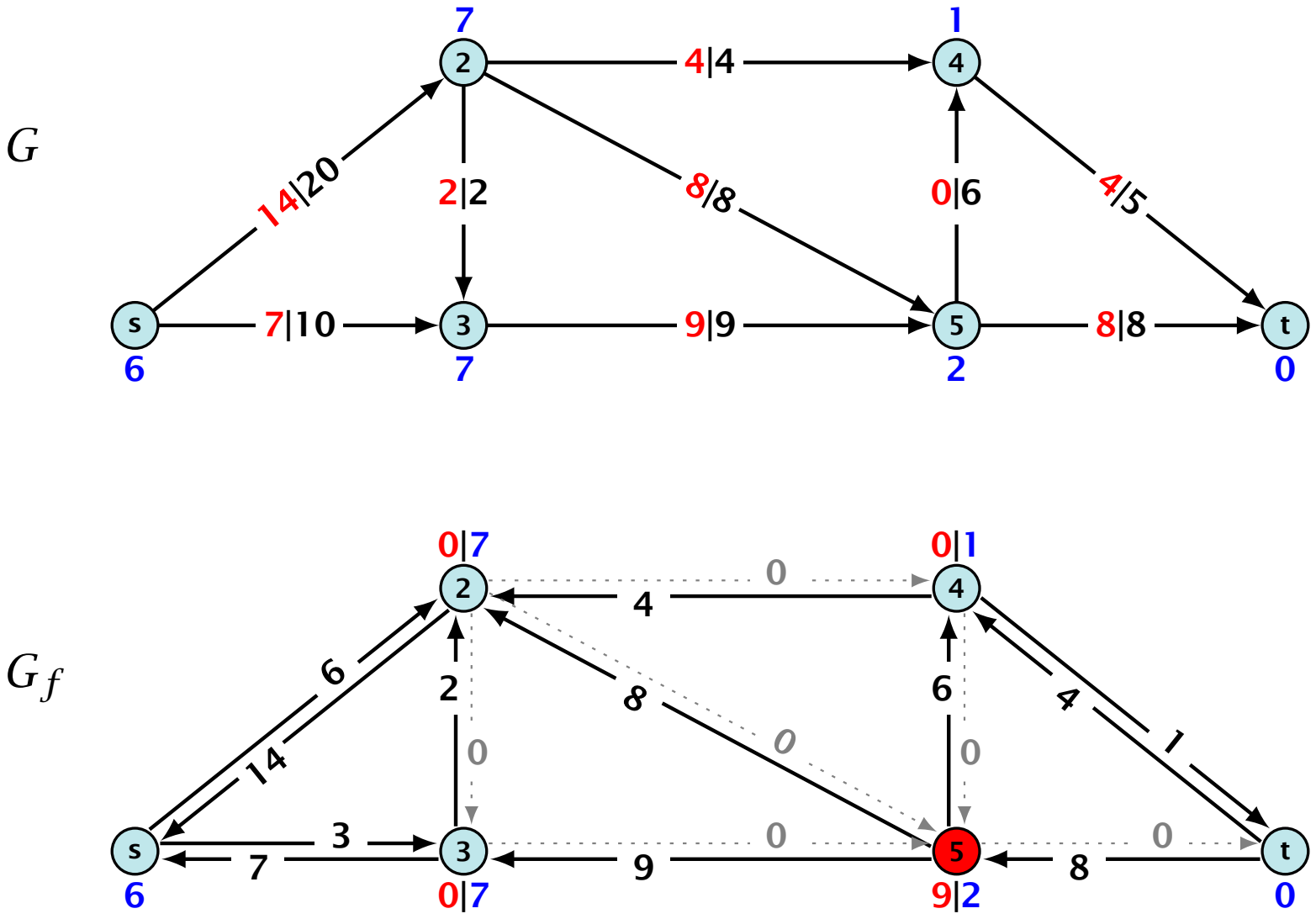


Preflow Push Algorithm

relabel

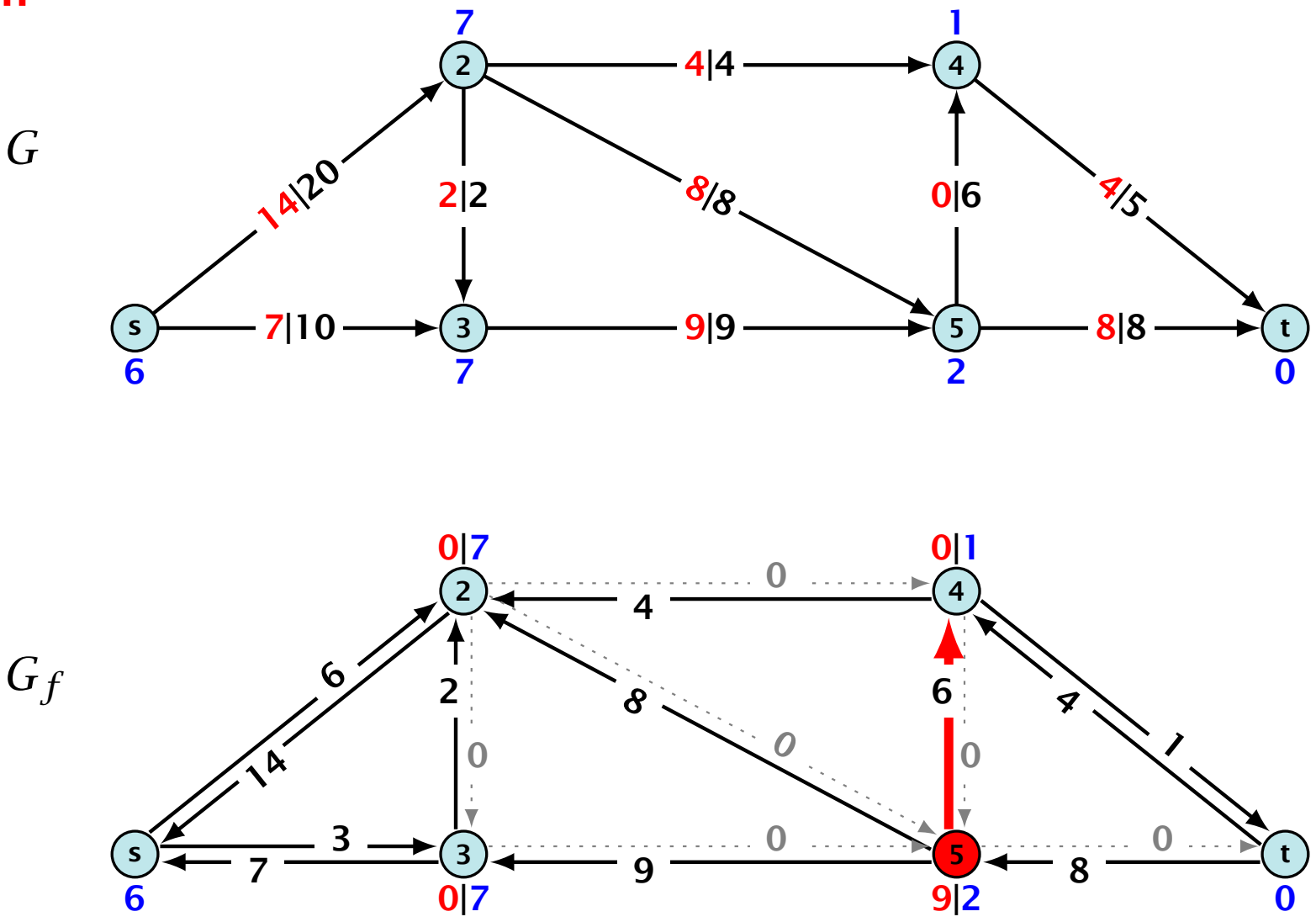


Preflow Push Algorithm

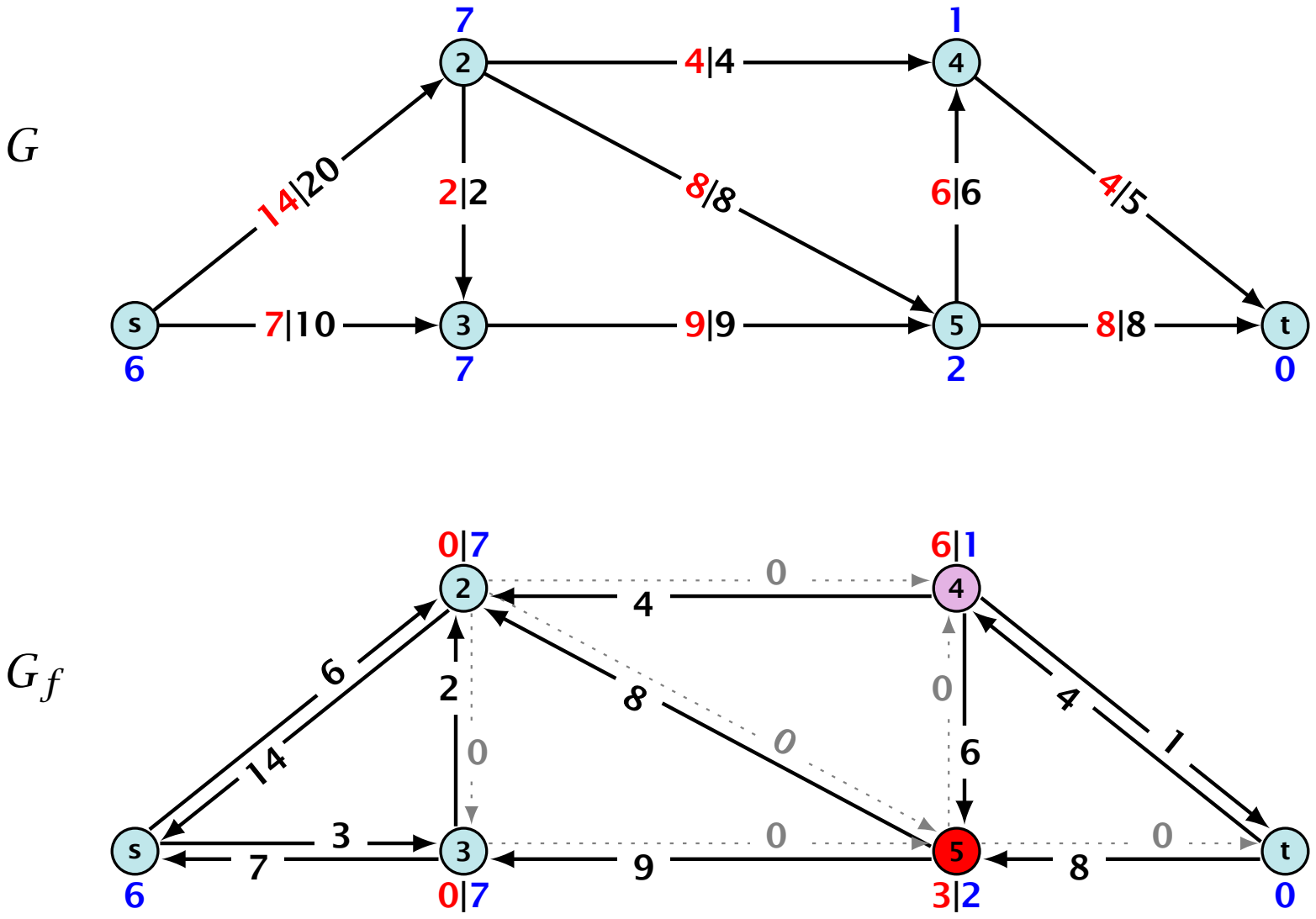


Preflow Push Algorithm

push

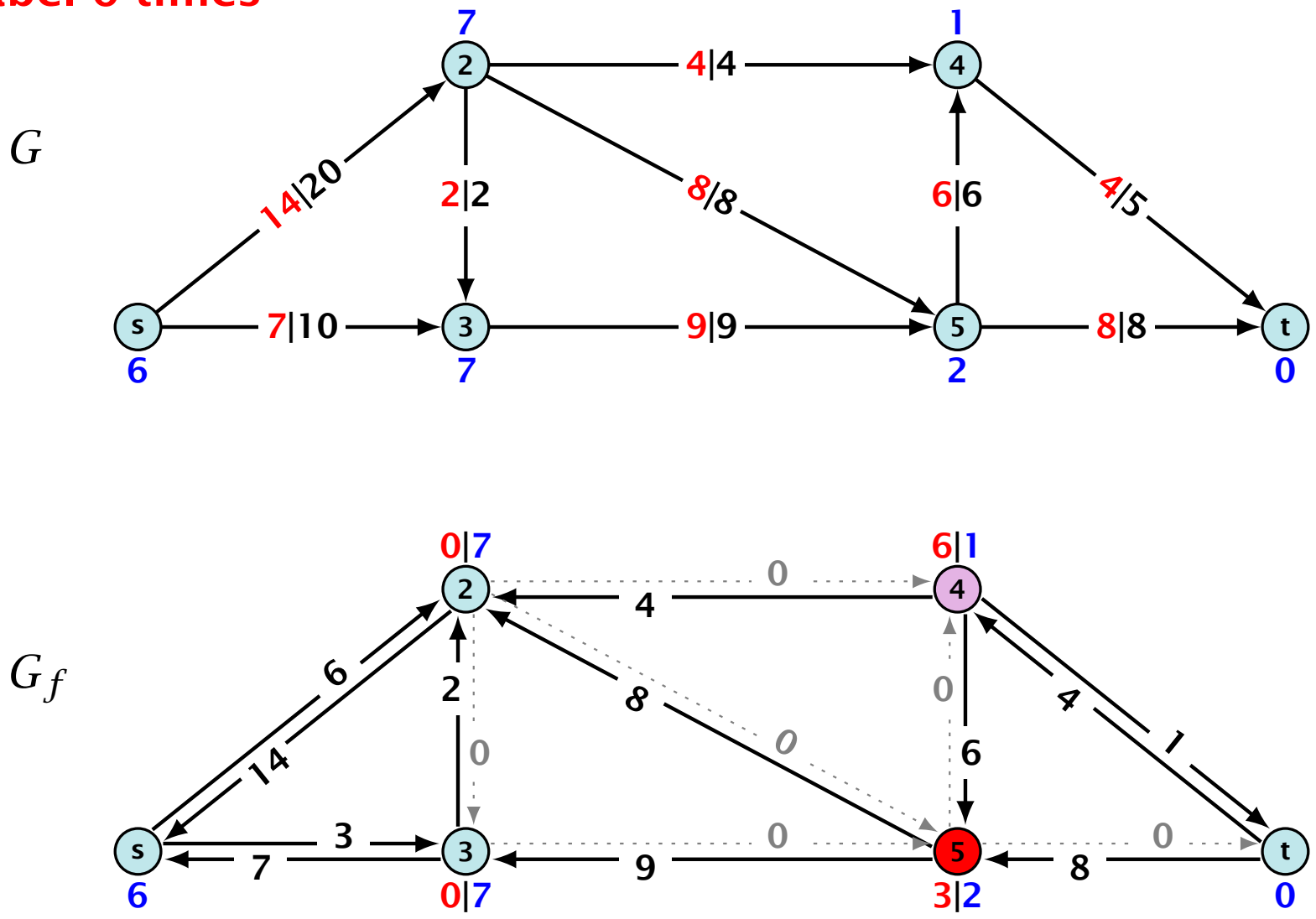


Preflow Push Algorithm

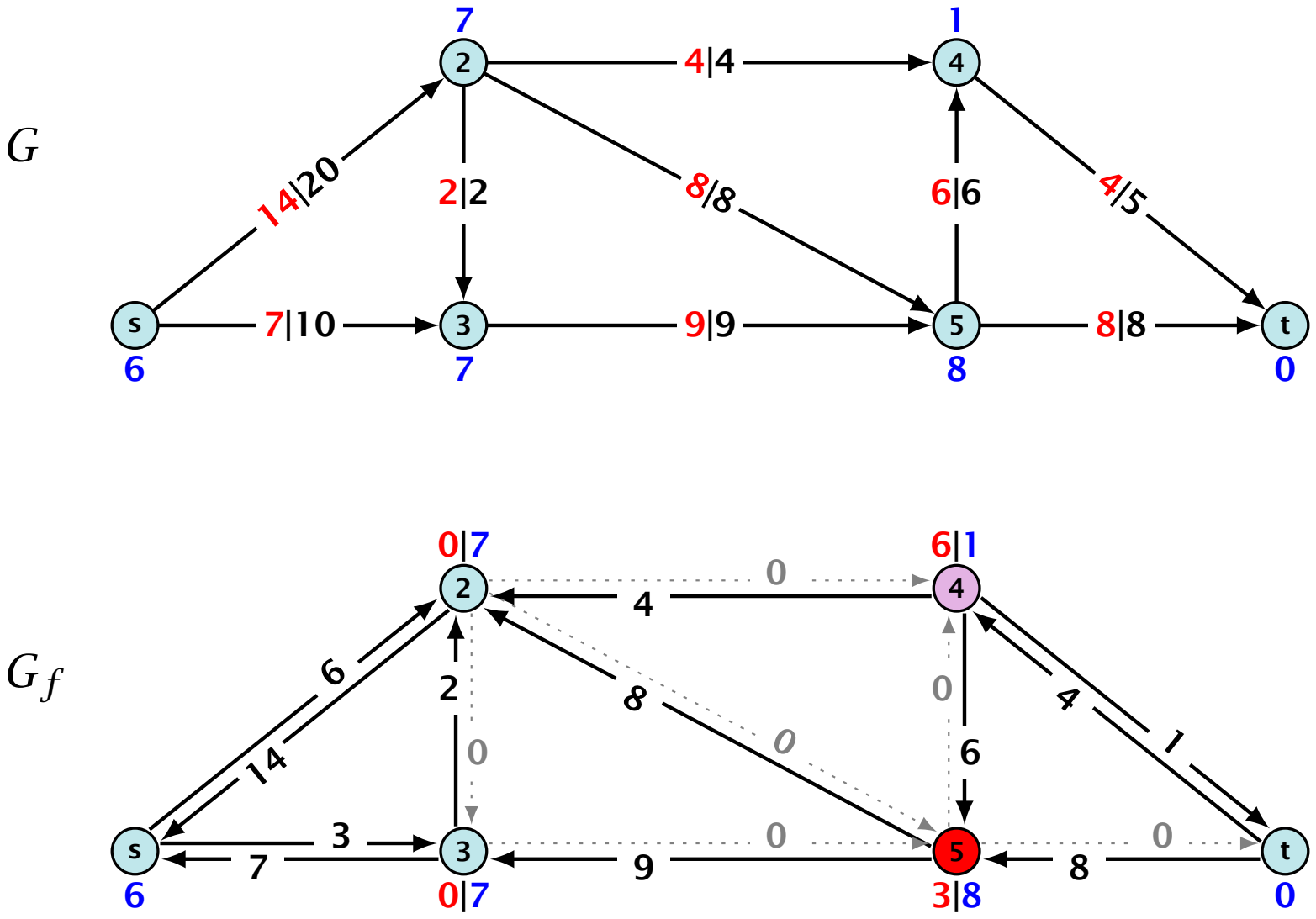


Preflow Push Algorithm

relabel 6 times

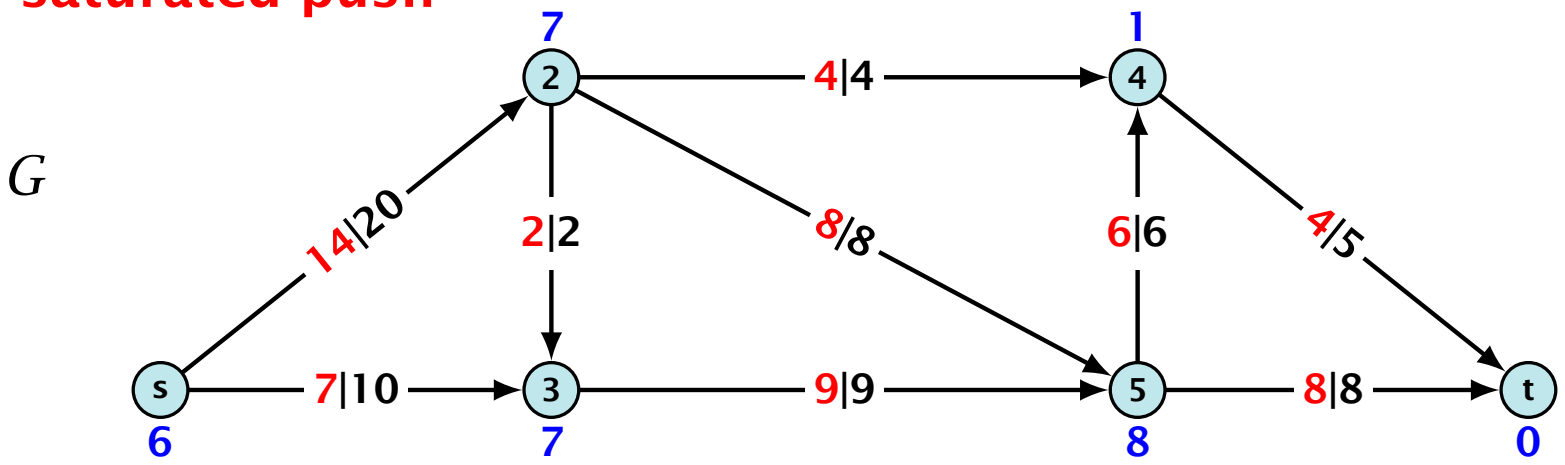


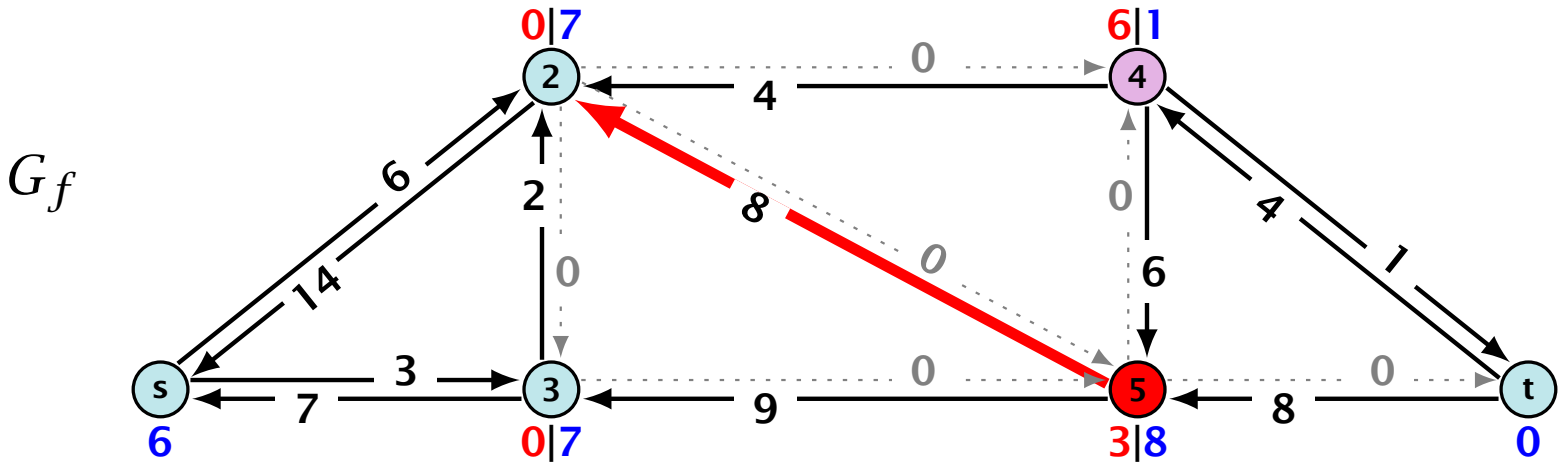
Preflow Push Algorithm



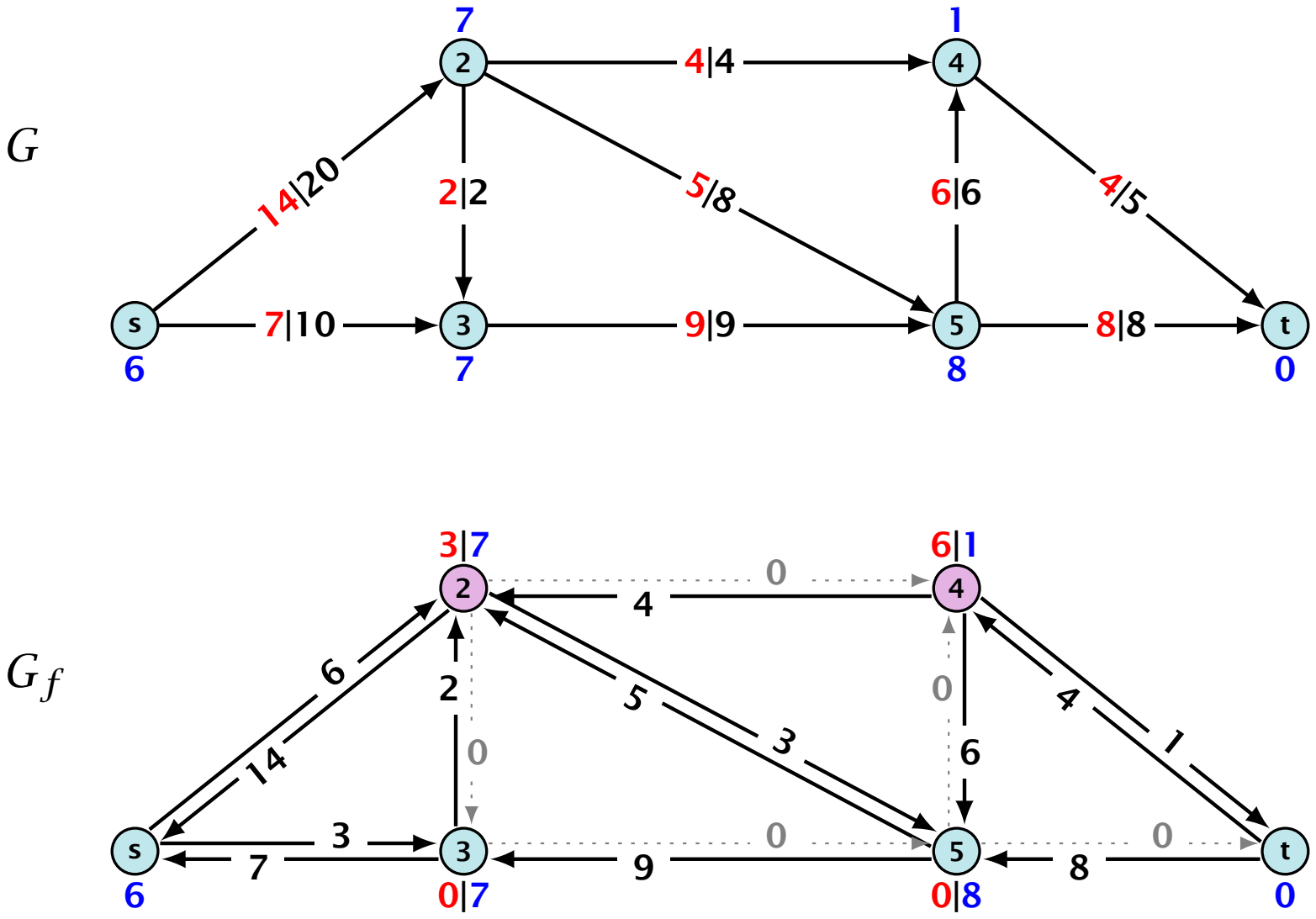
Preflow Push Algorithm

non-saturated push

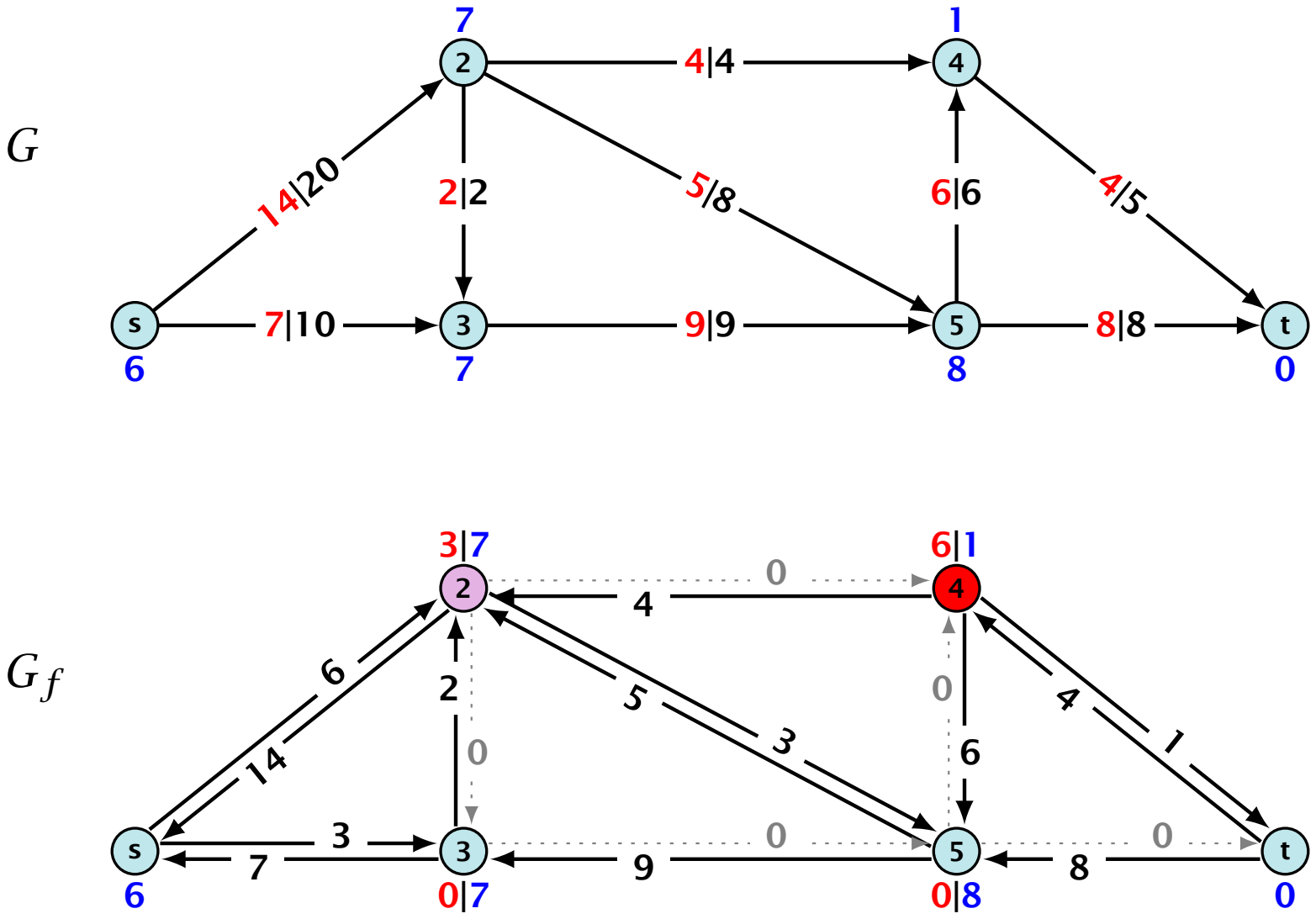




Preflow Push Algorithm

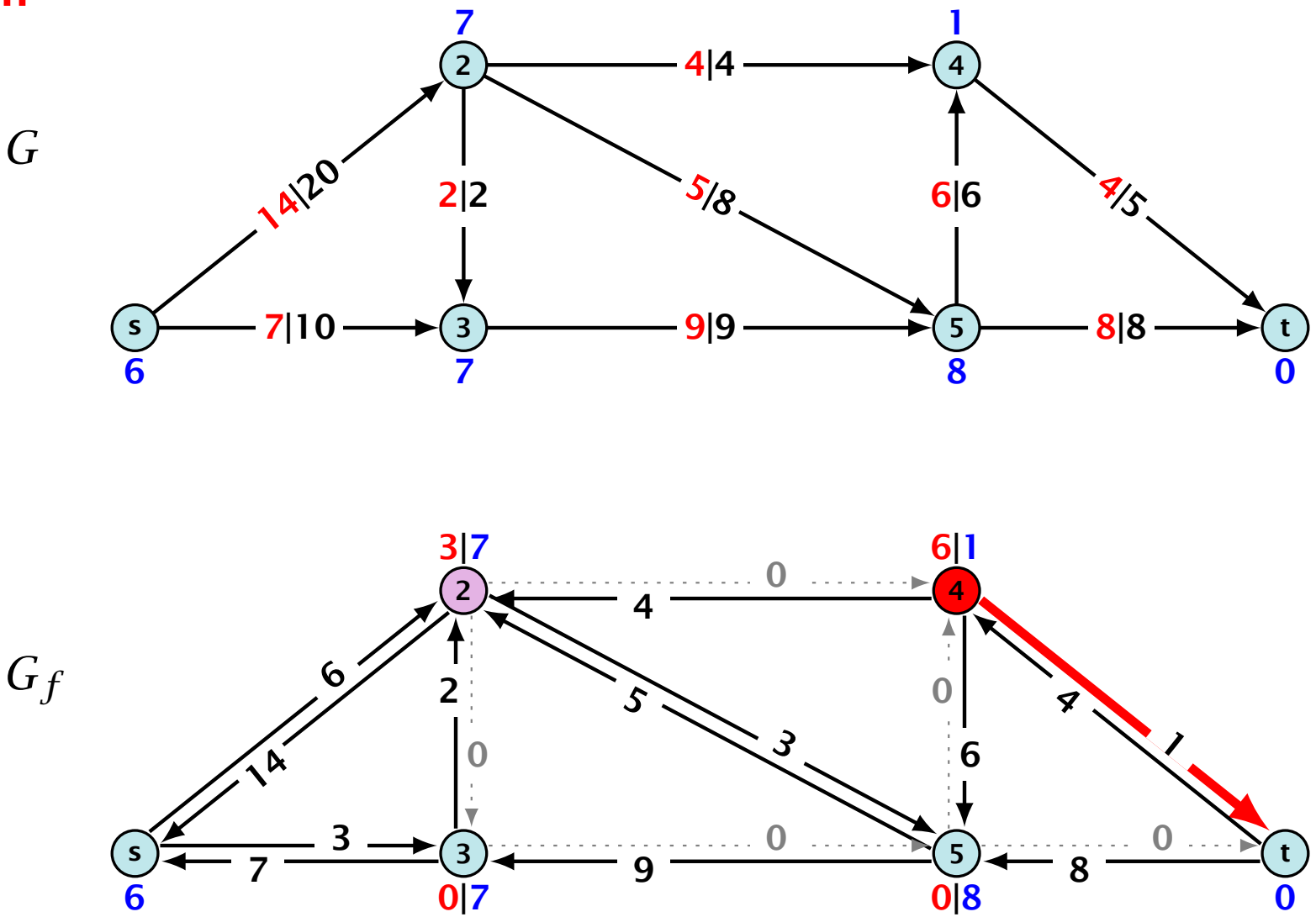


Preflow Push Algorithm

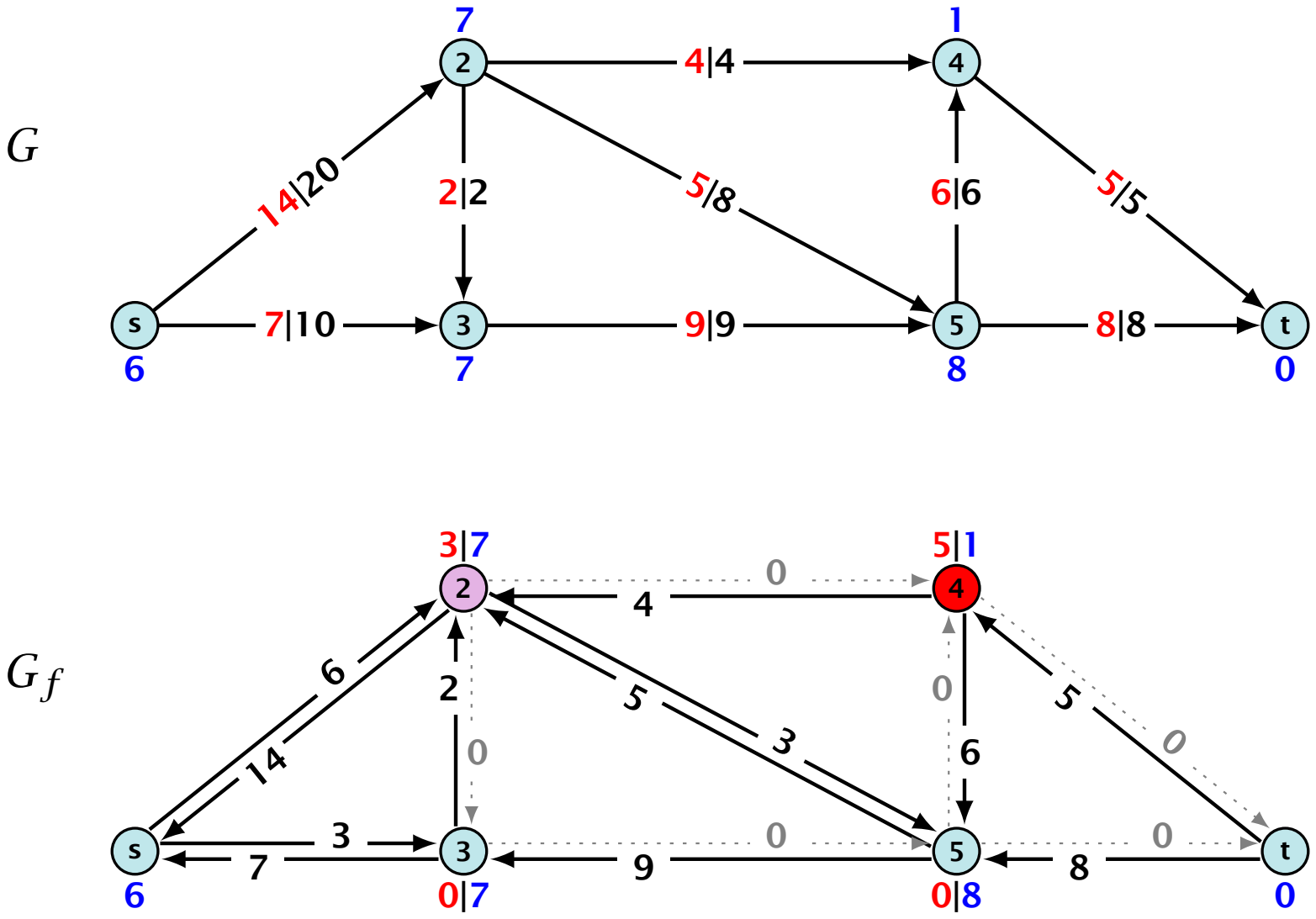


Preflow Push Algorithm

push

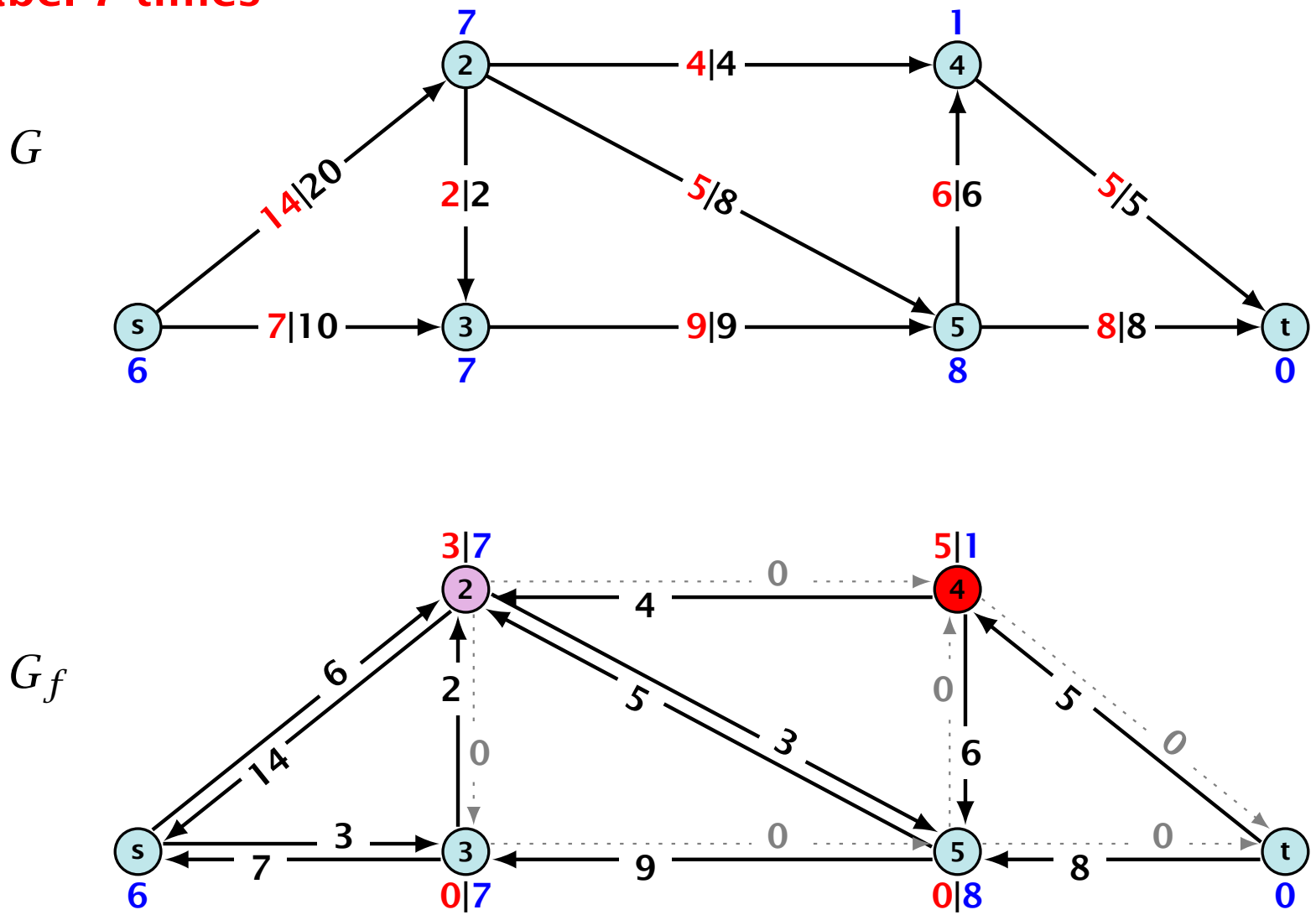


Preflow Push Algorithm

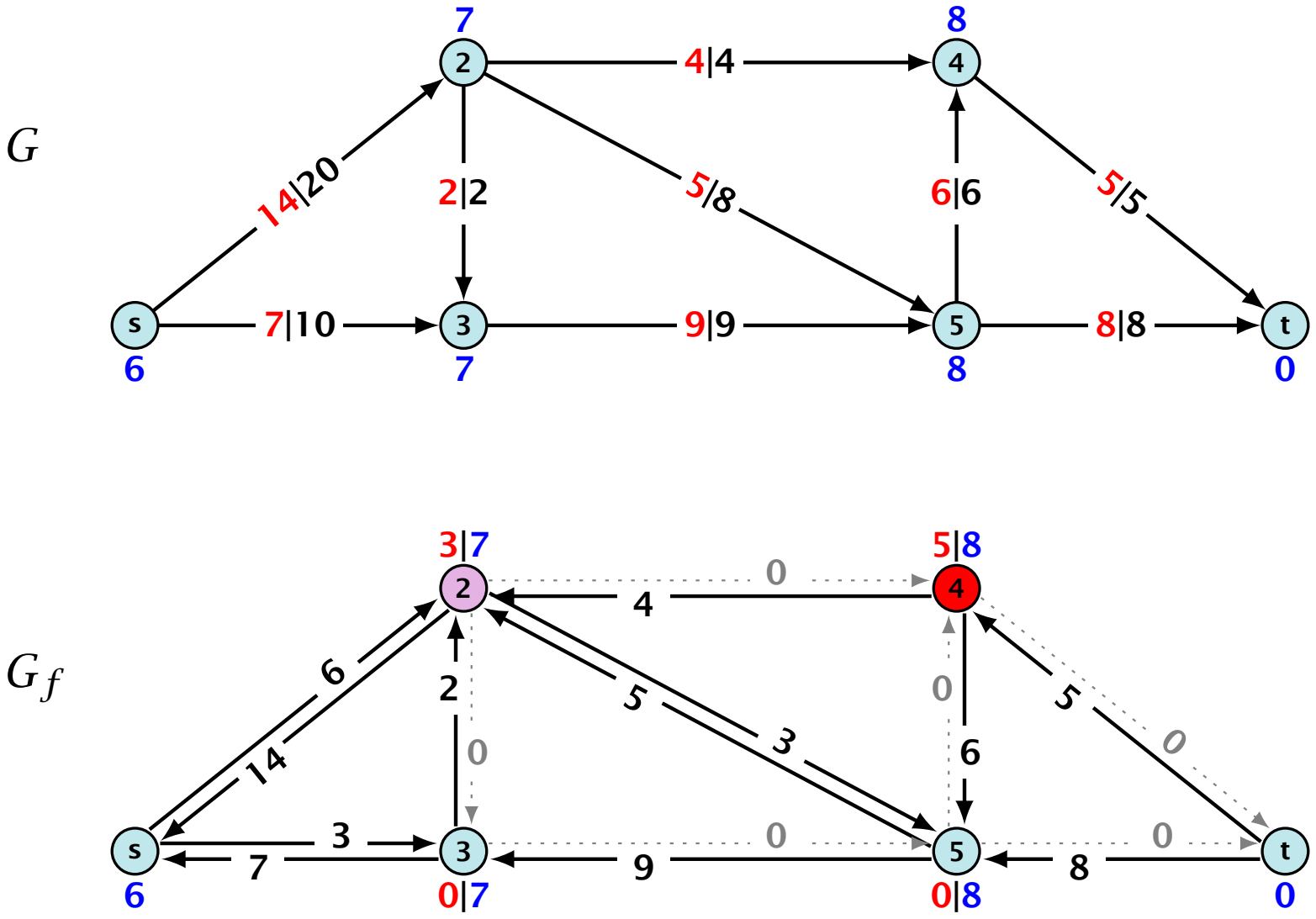


Preflow Push Algorithm

relabel 7 times

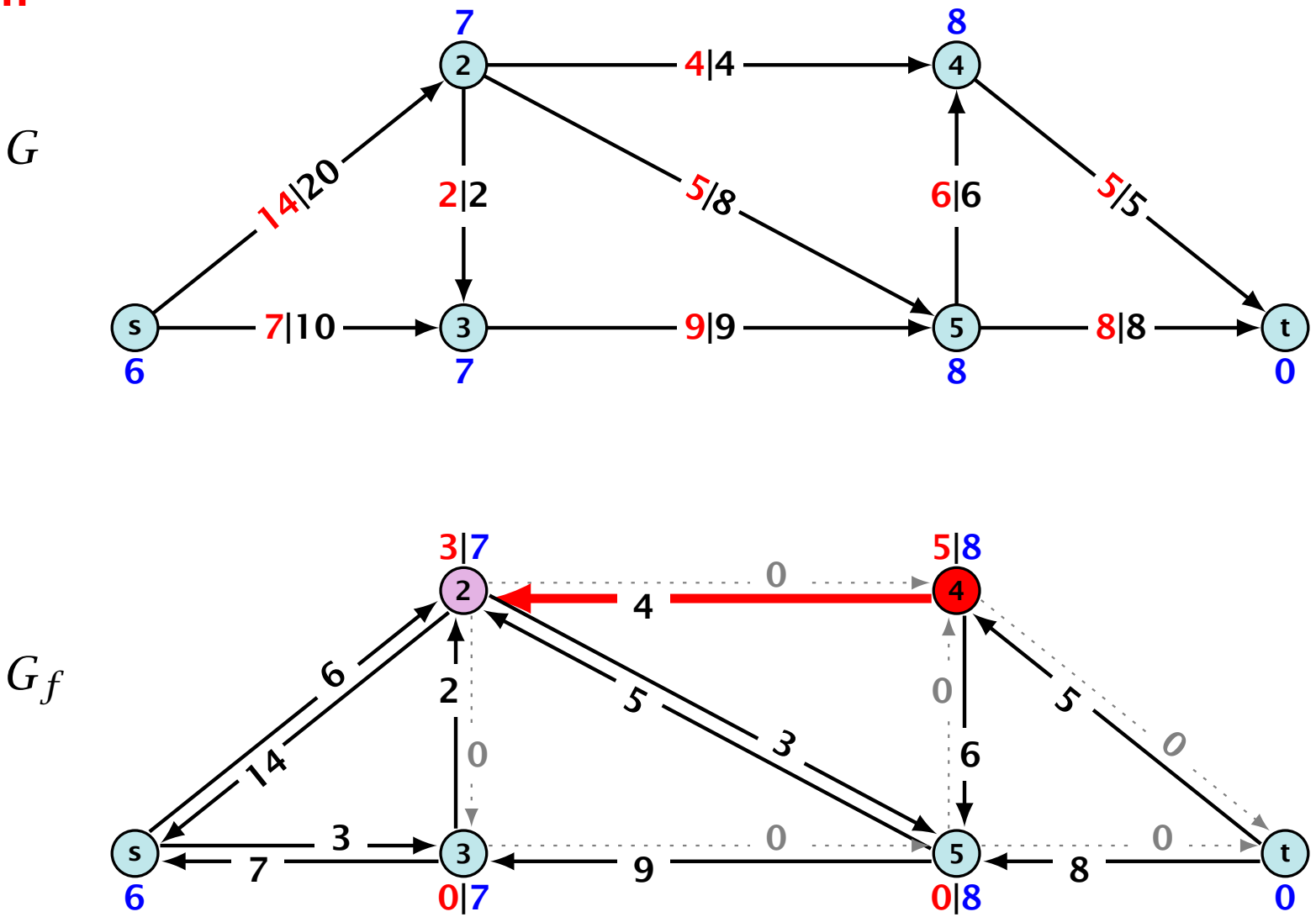


Preflow Push Algorithm

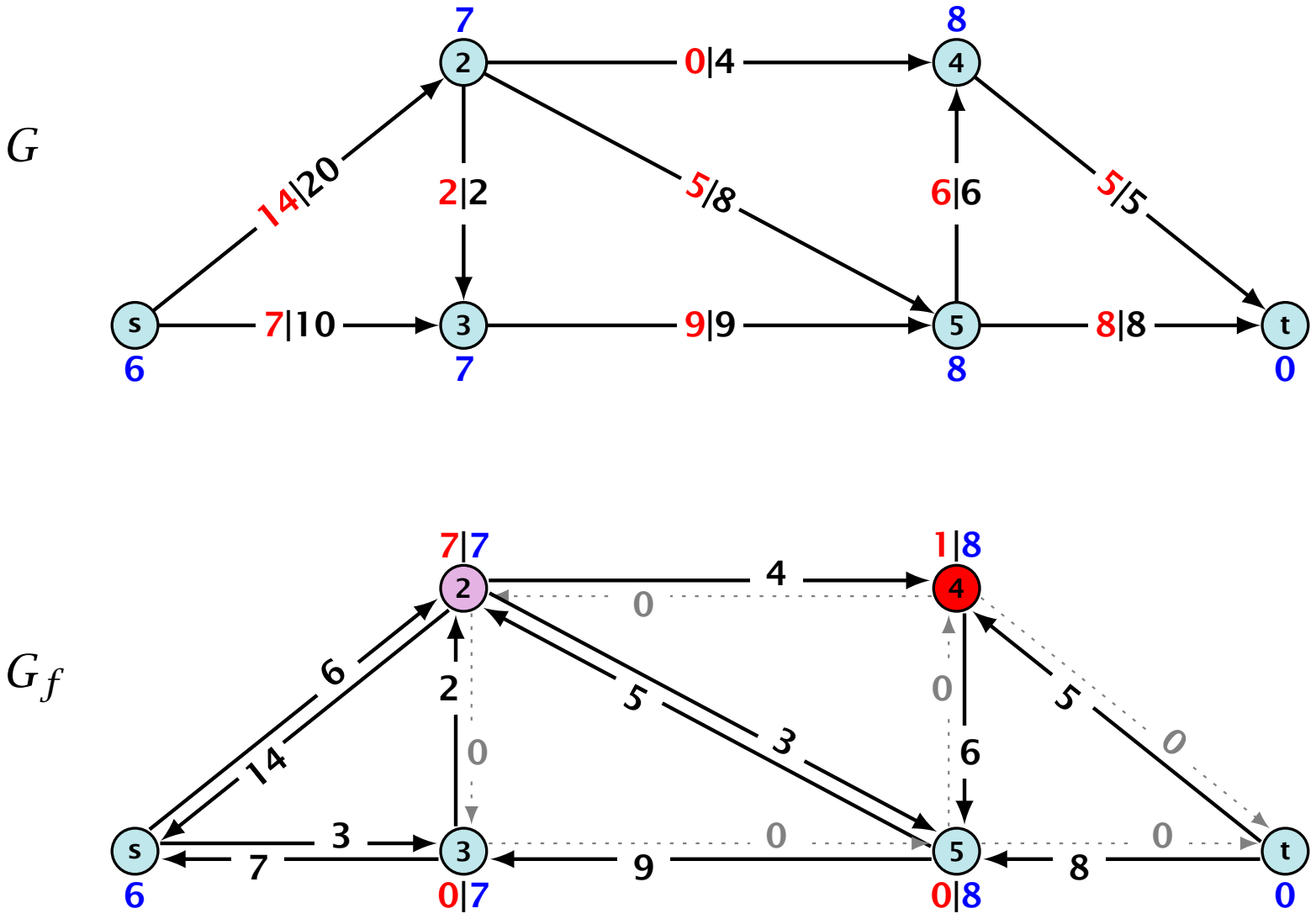


Preflow Push Algorithm

push

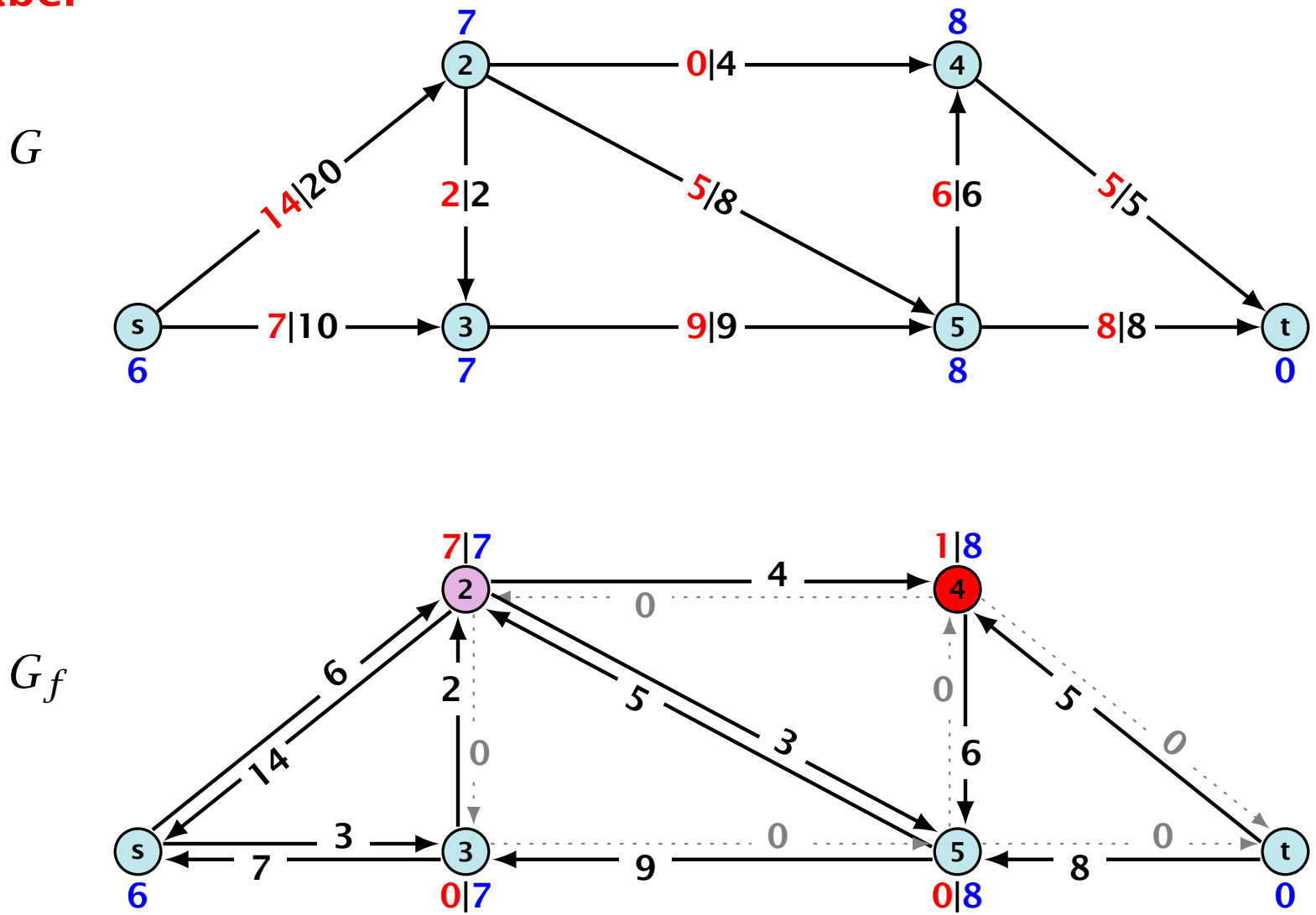


Preflow Push Algorithm

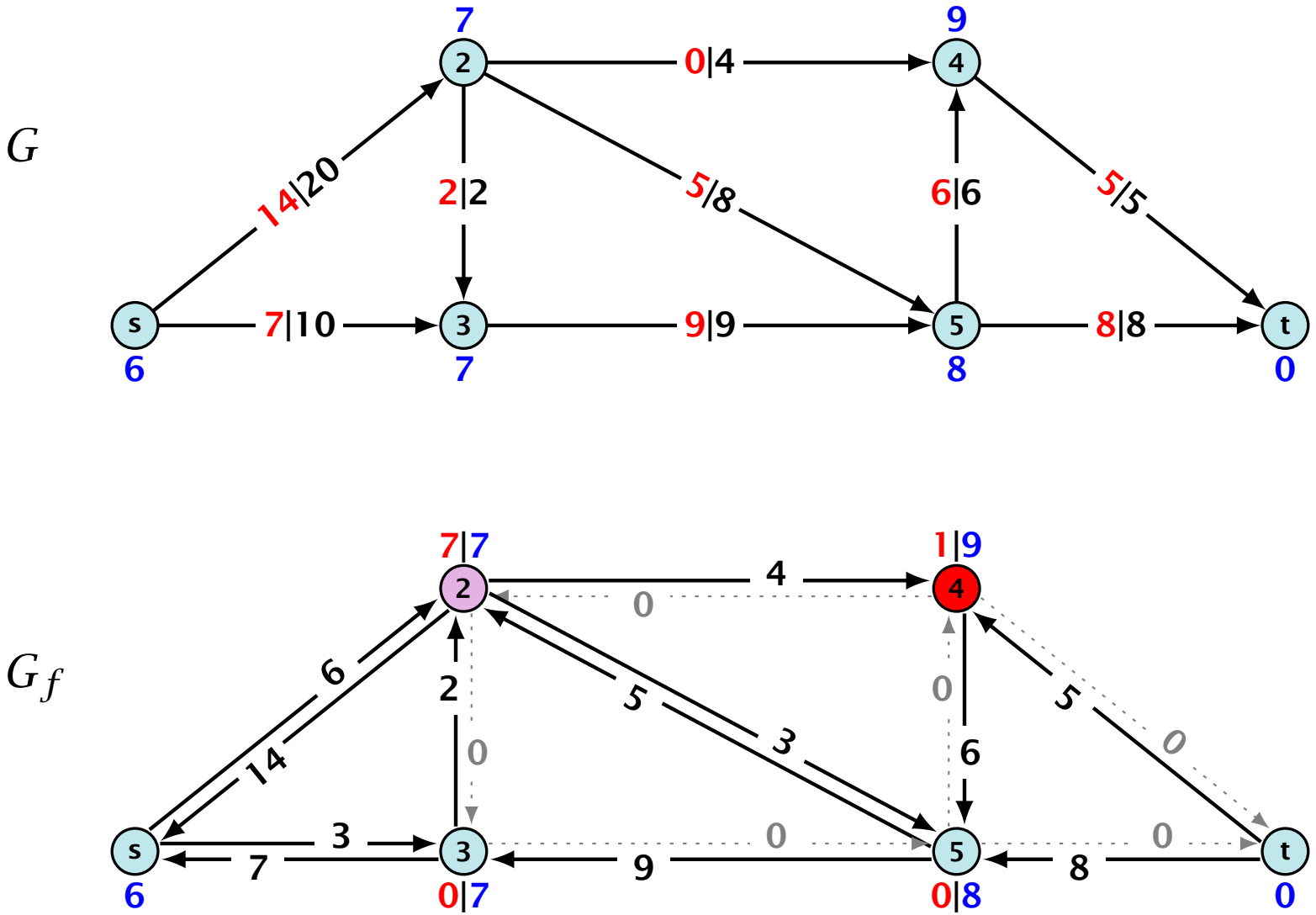


Preflow Push Algorithm

relabel

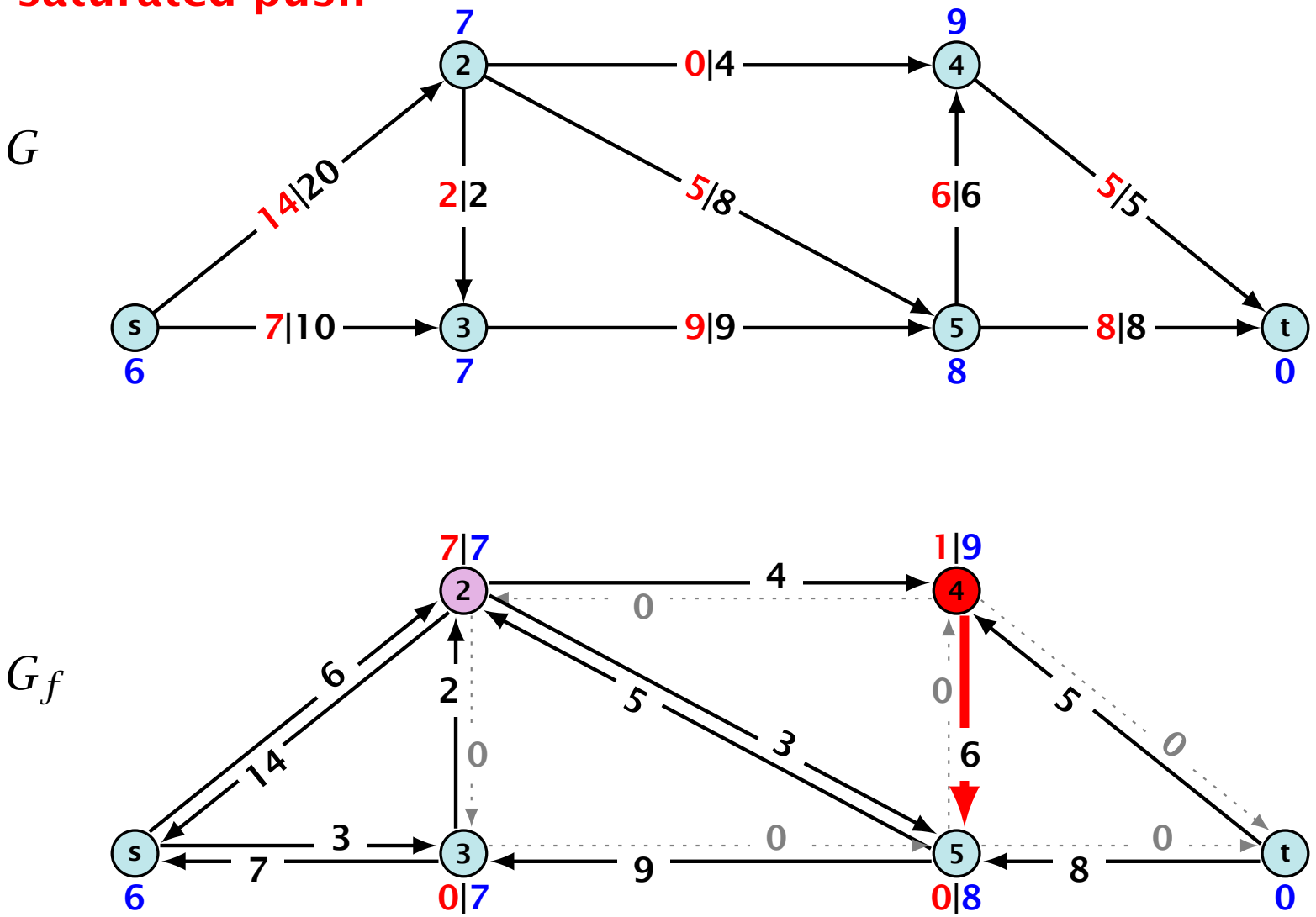


Preflow Push Algorithm

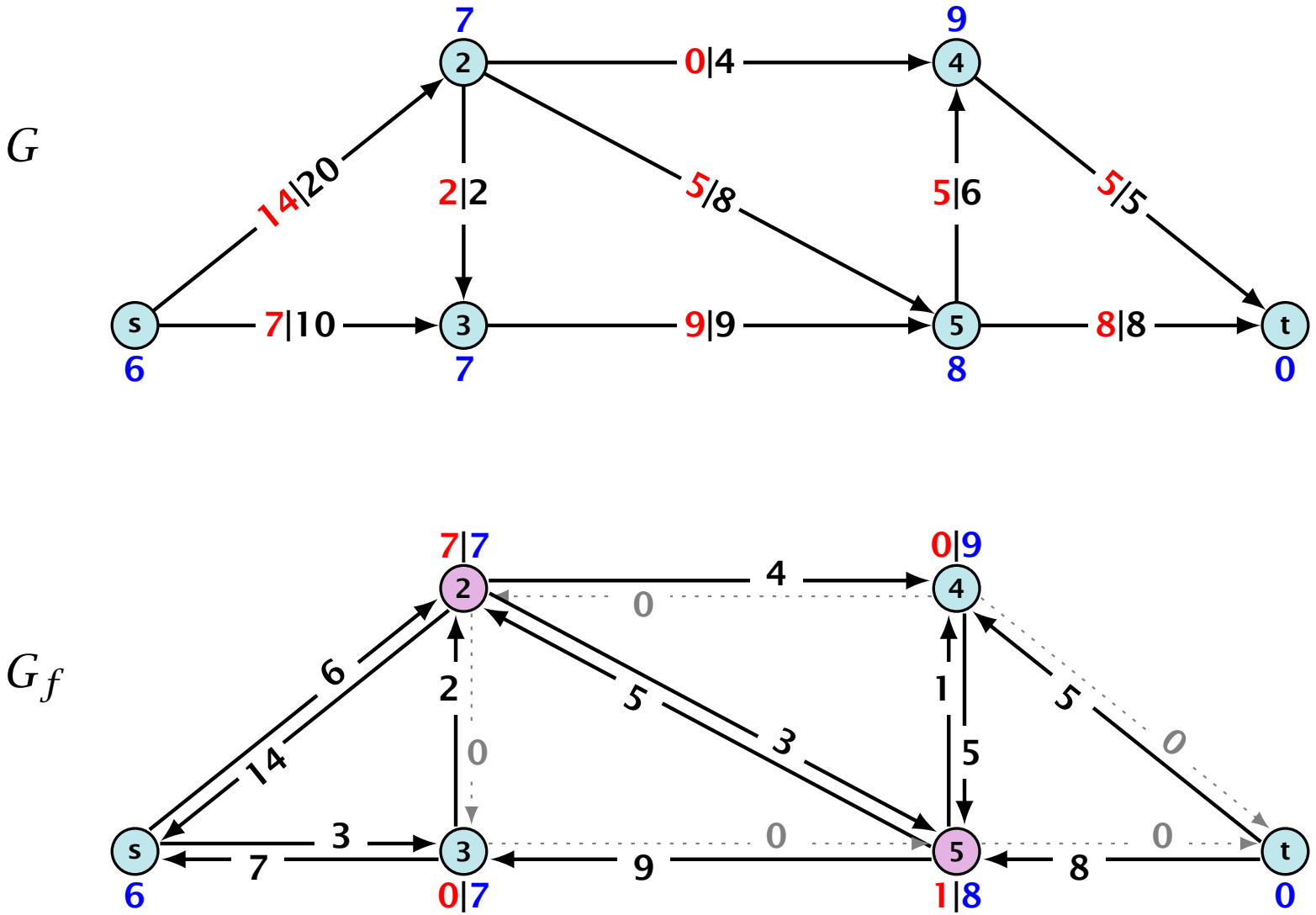


Preflow Push Algorithm

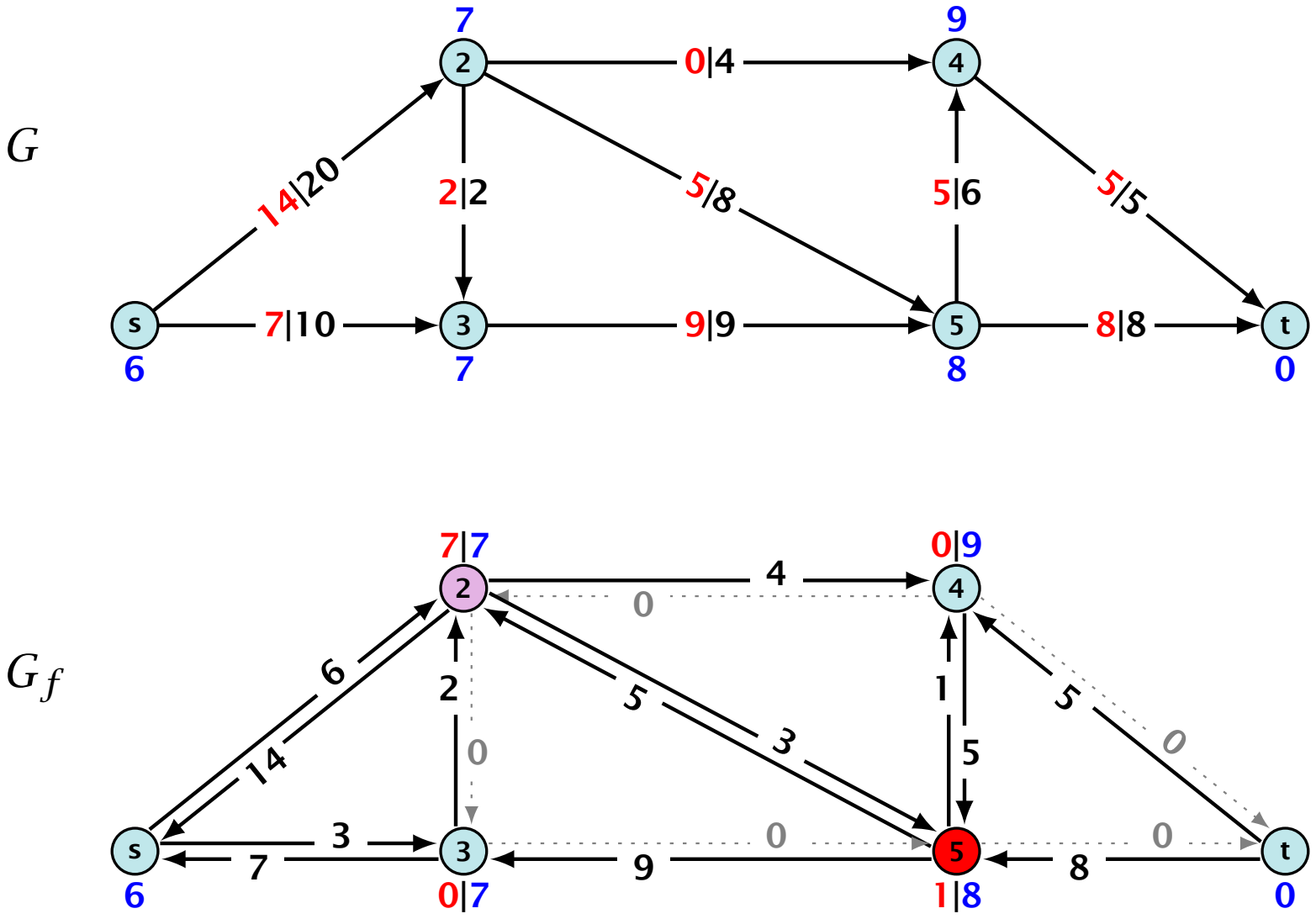
non-saturated push



Preflow Push Algorithm

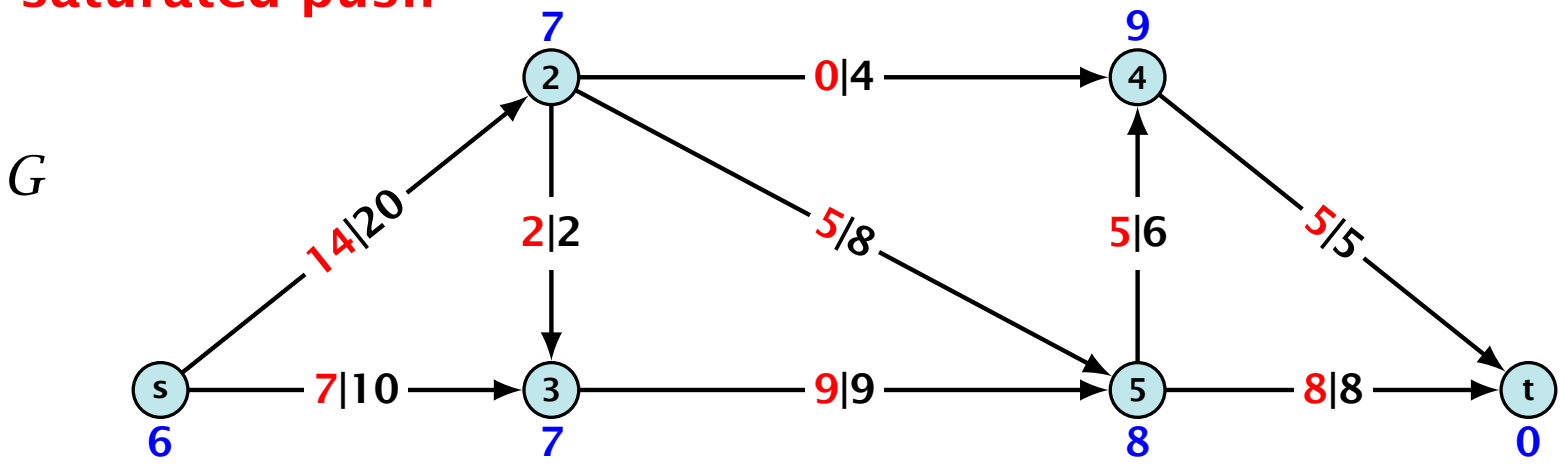


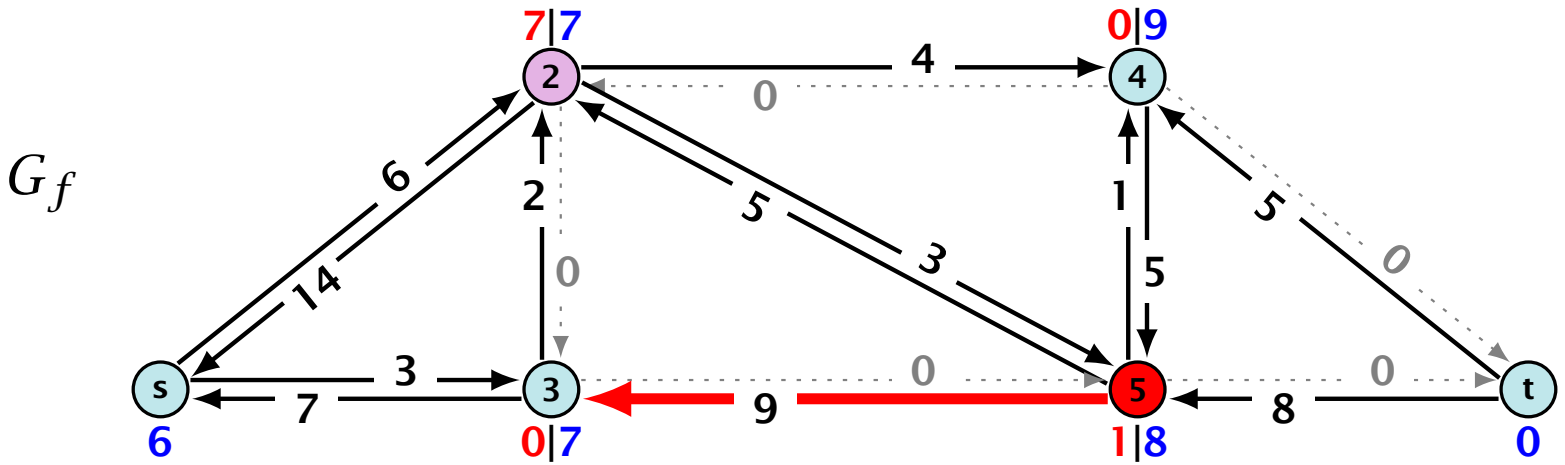
Preflow Push Algorithm



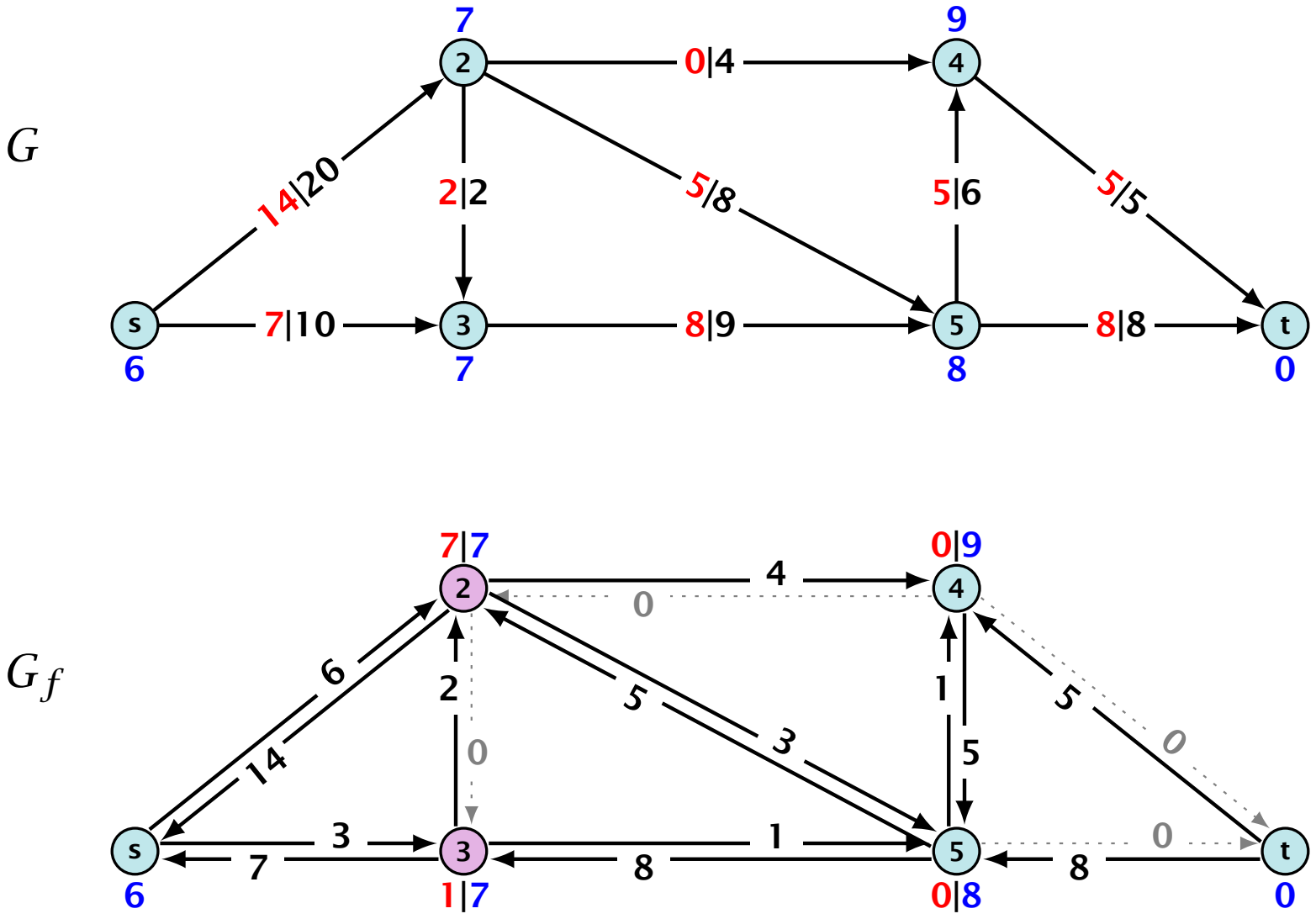
Preflow Push Algorithm

non-saturated push

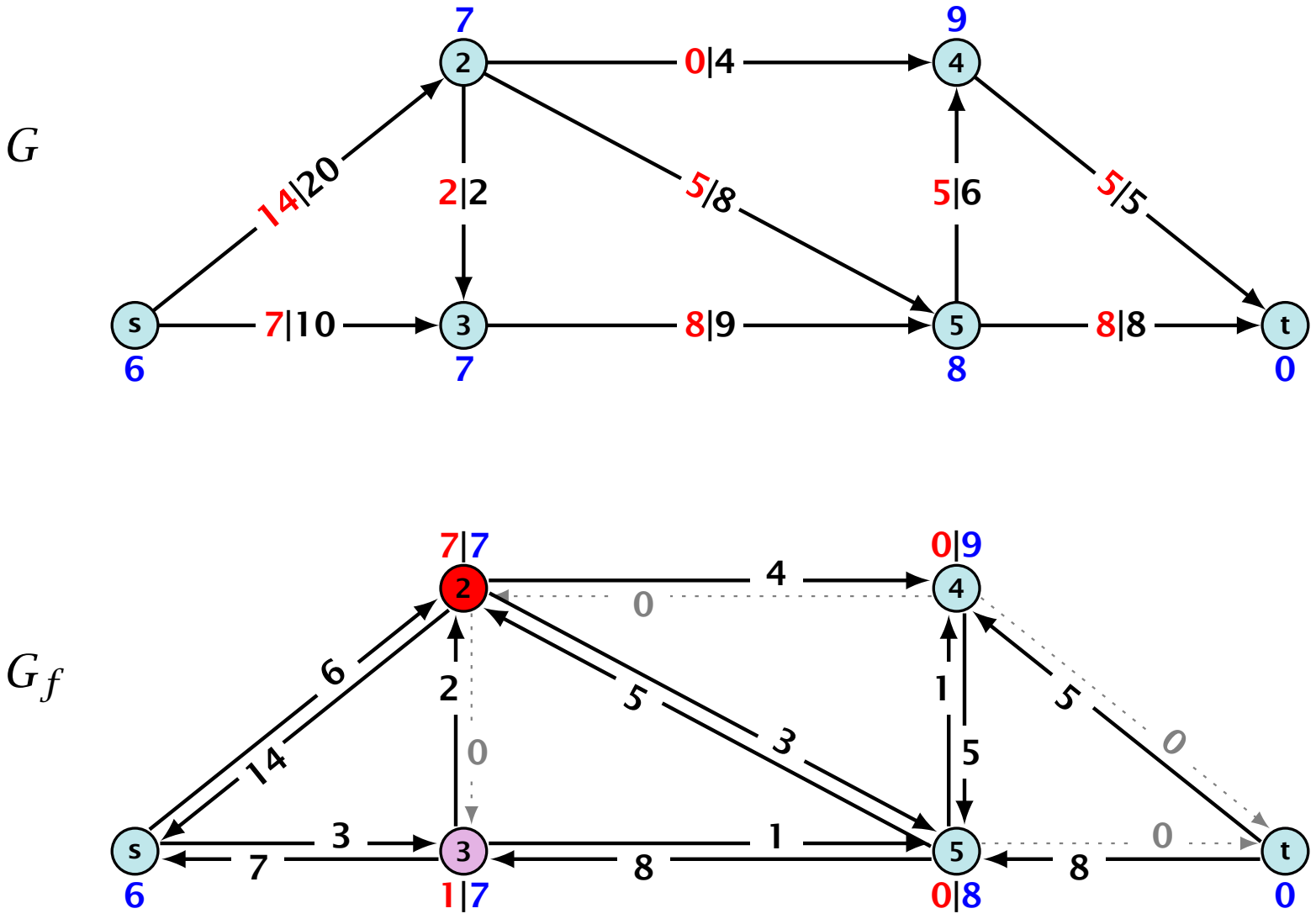




Preflow Push Algorithm

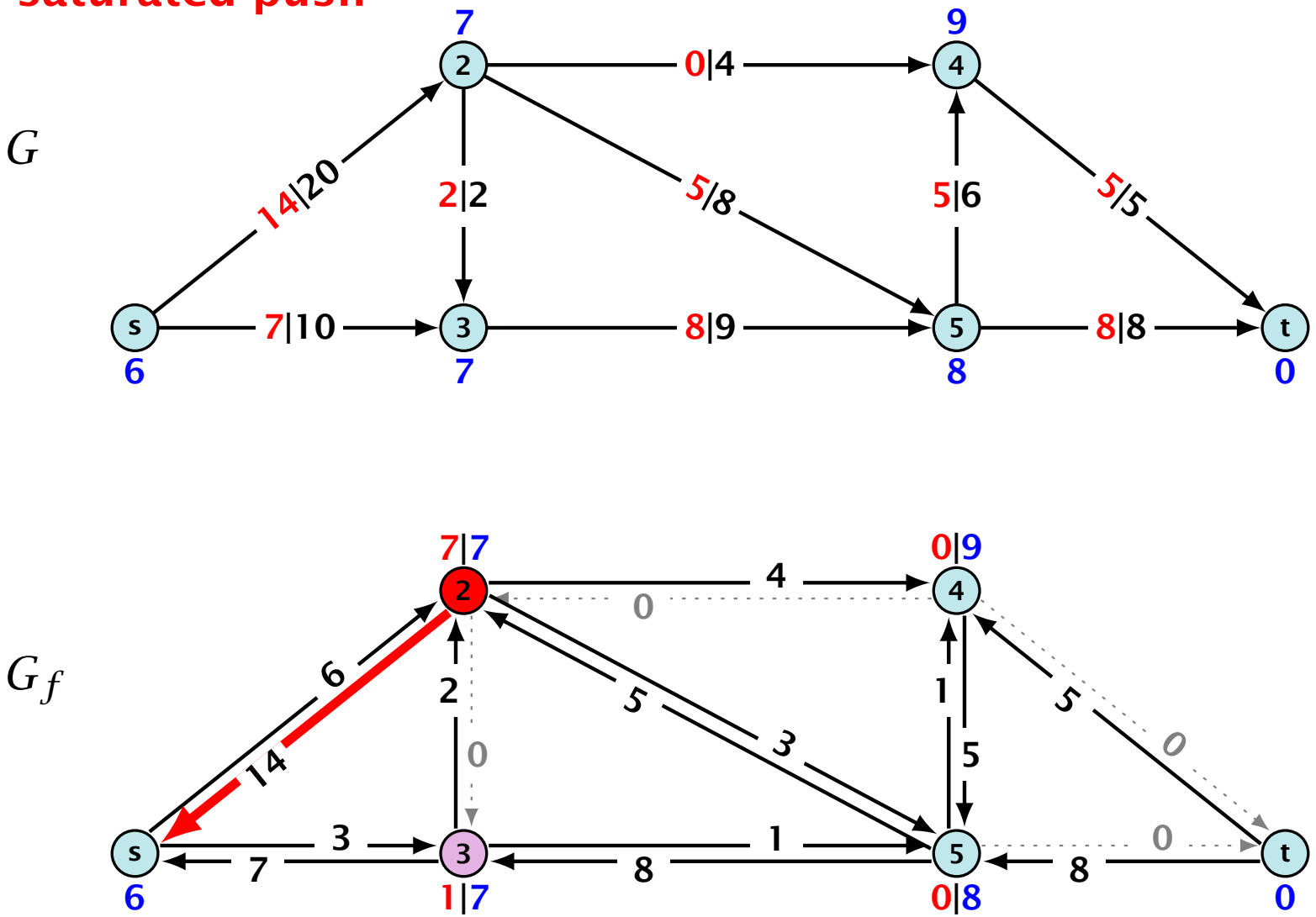


Preflow Push Algorithm

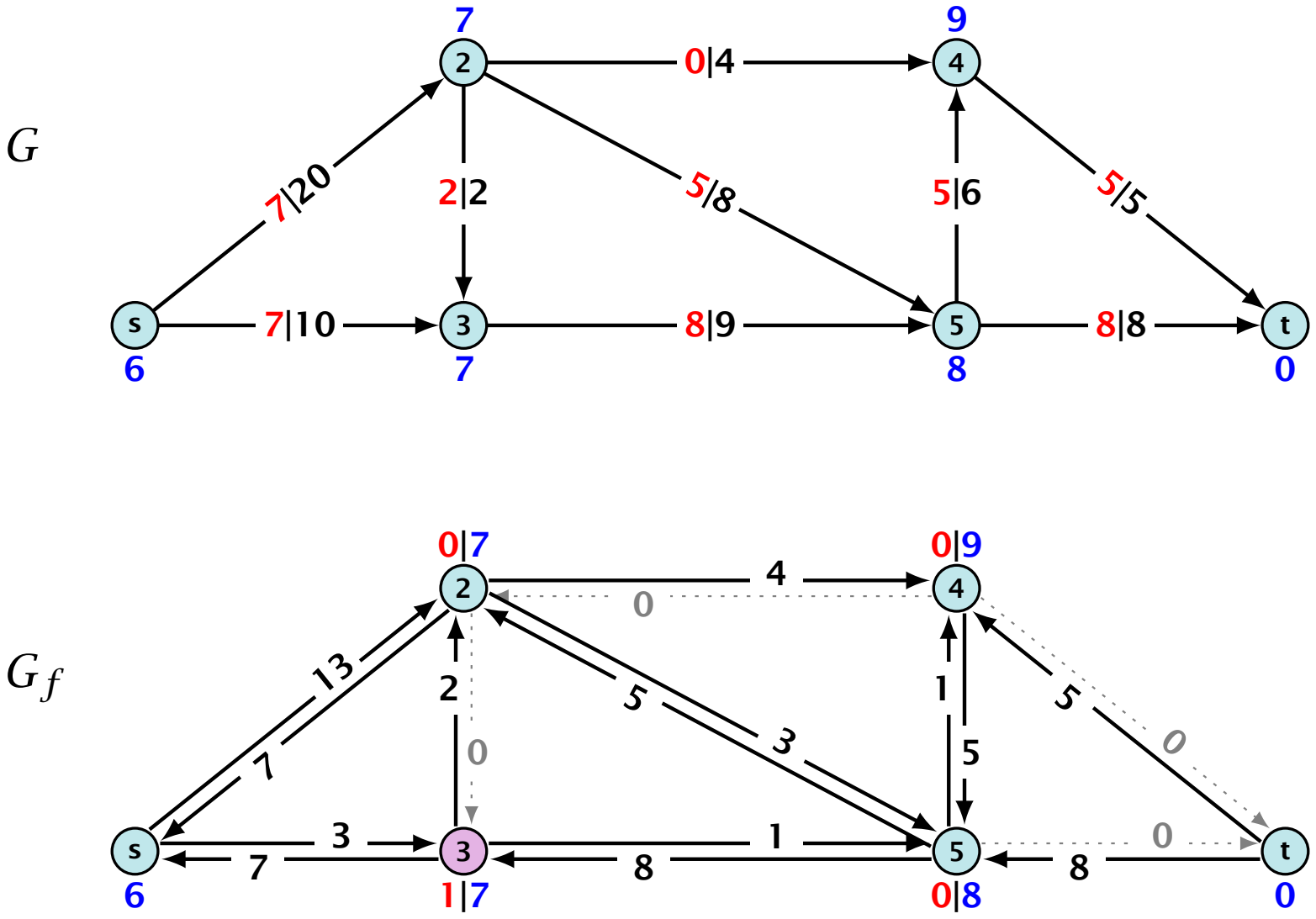


Preflow Push Algorithm

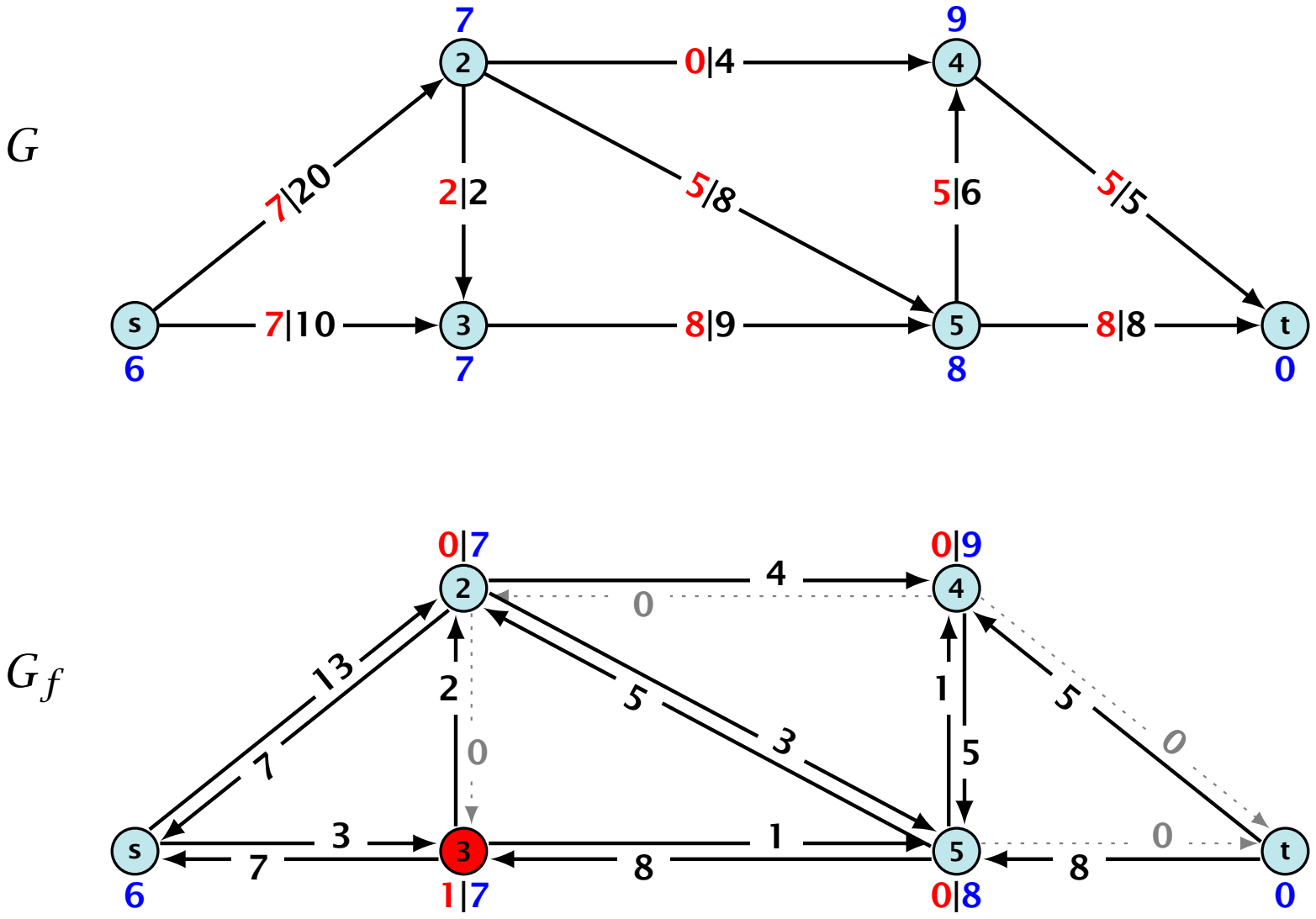
non-saturated push



Preflow Push Algorithm

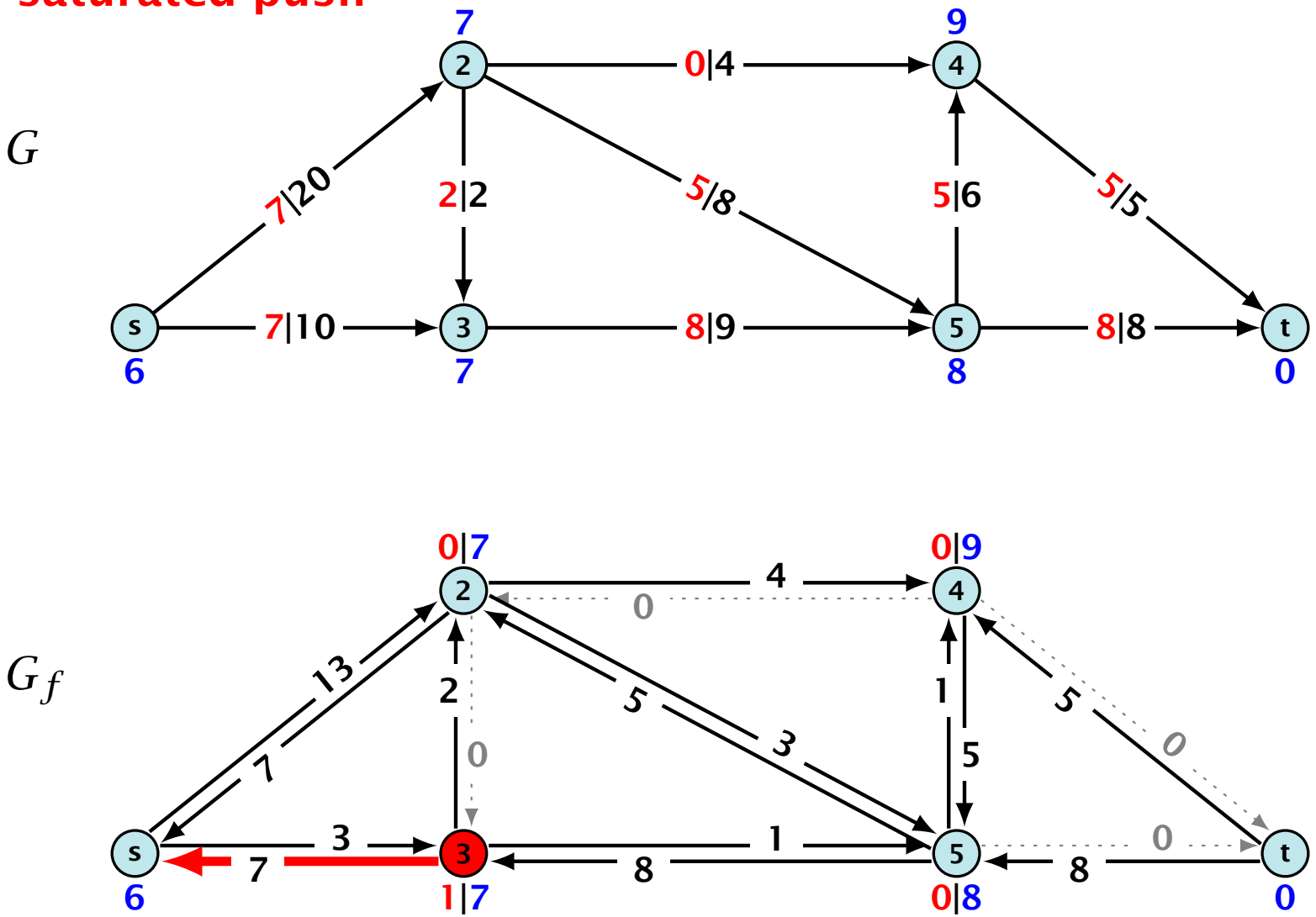


Preflow Push Algorithm

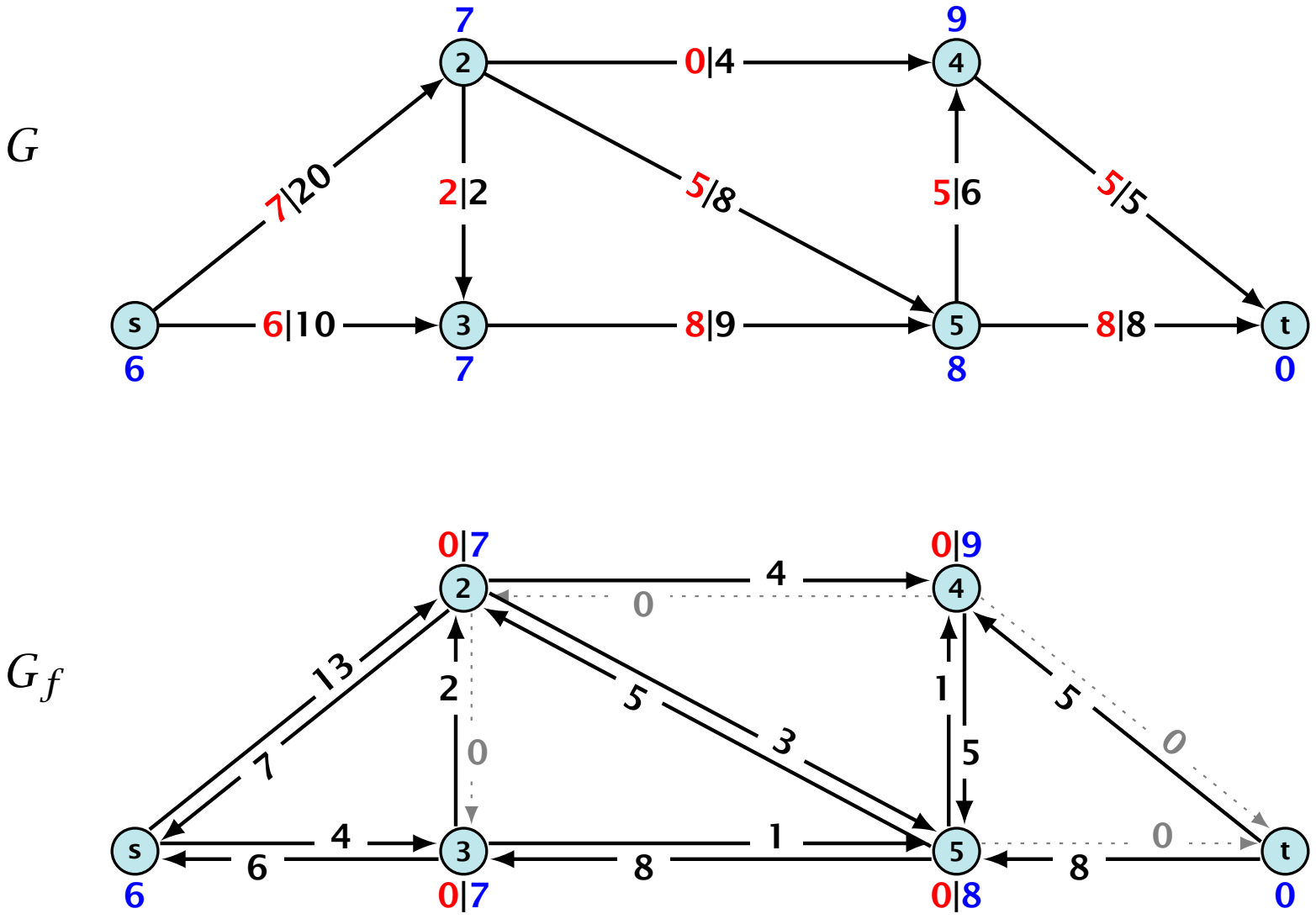


Preflow Push Algorithm

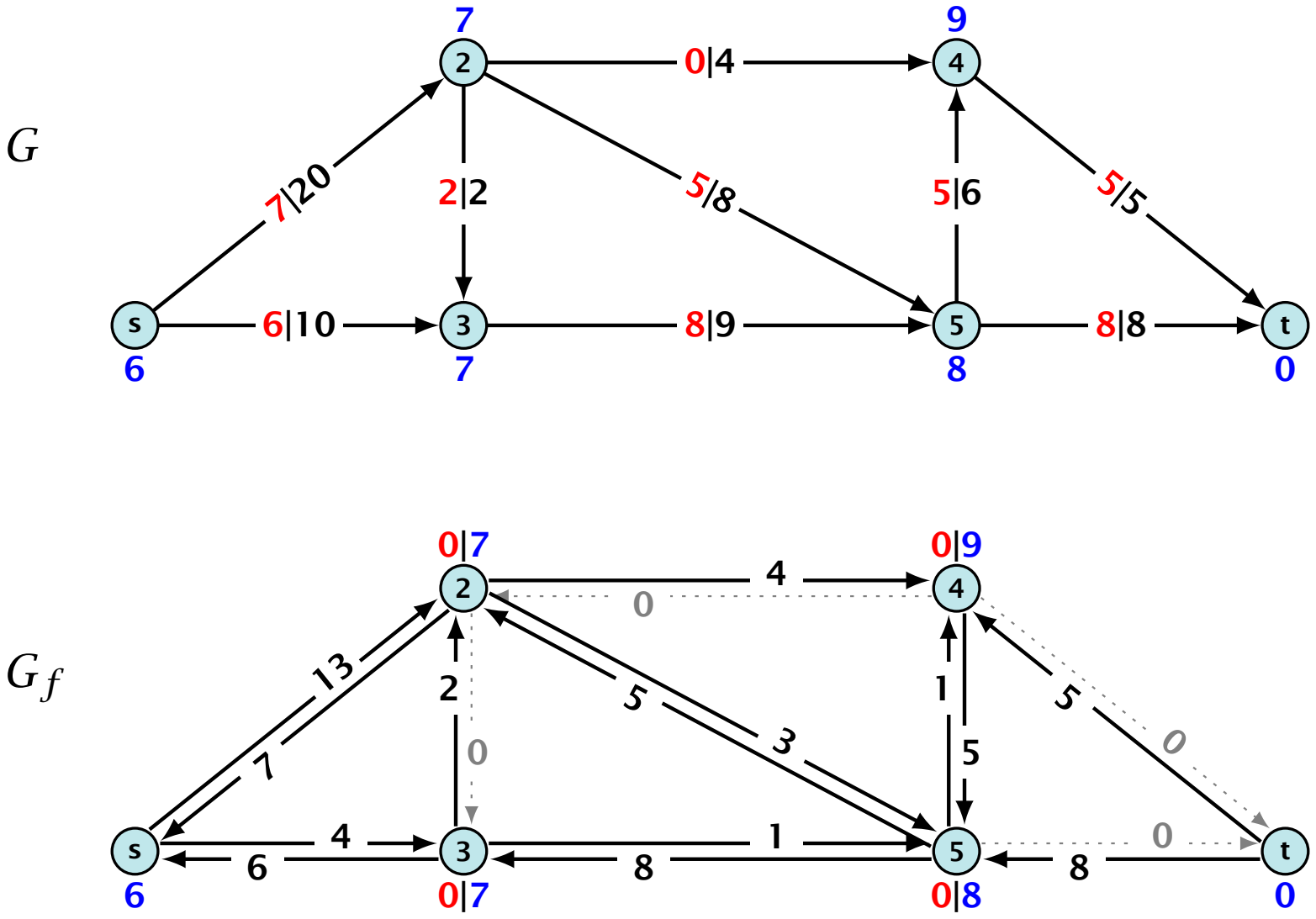
non-saturated push



Preflow Push Algorithm



Preflow Push Algorithm



Analysis

Lemma 68

An active node has a path to s in the residual graph.

Analysis

Lemma 68

An active node has a path to s in the residual graph.

Proof.

- Let A denote the set of nodes that can reach s, and let B denote the remaining nodes. Note that $s \in A$.

Analysis

Lemma 68

An active node has a path to s in the residual graph.

Proof.

- Let A denote the set of nodes that can reach s, and let B denote the remaining nodes. Note that $s \in A$.
- In the following we show that a node $b \in B$ has excess flow $f(b) = 0$ which gives the lemma.

Analysis

Lemma 68

An active node has a path to s in the residual graph.

Proof.

- Let A denote the set of nodes that can reach s, and let B denote the remaining nodes. Note that $s \in A$.
- In the following we show that a node $b \in B$ has excess flow $f(b) = 0$ which gives the lemma.
- In the residual graph there are no edges into A, and, hence, no edges leaving A/entering B can carry any flow.

Analysis

Lemma 68

An active node has a path to s in the residual graph.

Proof.

- Let A denote the set of nodes that can reach s, and let B denote the remaining nodes. Note that $s \in A$.
- In the following we show that a node $b \in B$ has excess flow $f(b) = 0$ which gives the lemma.
- In the residual graph there are no edges into A, and, hence, no edges leaving A/entering B can carry any flow.
- Let $f(B) = \sum_{v \in B} f(v)$ be the excess flow of all nodes in B.

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$f(B)$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$f(B) = \sum_{b \in B} f(b)$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= \sum_{b \in B} \sum_{v \in A} f(v,b) - \sum_{b \in B} \sum_{v \in A} f(b,v) + \sum_{b \in B} \sum_{v \in B} f(v,b) - \sum_{b \in B} \sum_{v \in B} f(b,v) \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= \sum_{b \in B} \sum_{v \in A} f(v,b) - \sum_{b \in B} \sum_{v \in A} f(b,v) + \underbrace{\sum_{b \in B} \sum_{v \in B} f(v,b) - \sum_{b \in B} \sum_{v \in B} f(b,v)}_{= 0} \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= \sum_{b \in B} \sum_{v \in A} f(v,b) - \sum_{b \in B} \sum_{v \in A} f(b,v) \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= \sum_{b \in B} \sum_{v \in A} \underbrace{f(v,b)}_{=0} - \sum_{b \in B} \sum_{v \in A} f(b,v) \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= \sum_{b \in B} \sum_{v \in A} f(v,b) - \sum_{b \in B} \sum_{v \in A} f(b,v) \\ &= 0 \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= - \sum_{b \in B} \sum_{v \in A} f(b,v) \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= - \sum_{b \in B} \sum_{v \in A} \underbrace{f(b,v)}_{\geq 0} \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= - \sum_{b \in B} \sum_{v \in A} f(b,v) \\ &\leq 0 \end{aligned}$$

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \left(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \right) \\ &= \sum_{b \in B} \left(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \right) \\ &= - \sum_{b \in B} \sum_{v \in A} f(b,v) \\ &\leq 0 \end{aligned}$$

Hence, the excess flow $f(b)$ must be 0 for every node $b \in B$.

Analysis

Lemma 69

The label of a node cannot become larger than $2n - 1$.

Proof.

- When increasing the label at a node u there exists a path from u to s of length at most $n - 1$. Along each edge of the path the height/label can at most drop by 1, and the label of the source is n.

Analysis

Lemma 70

There are only $\mathcal{O}(n^3)$ calls to discharge when using the relabel-to-front heuristic.

Proof.

- When increasing the label at a node u there exists a path from u to s of length at most $n-1$. Along each edge of the path the height/label can at most drop by 1, and the label of the source is n.

Lemma 71

The number of saturating pushes performed is at most $\mathcal{O}(mn)$.

Lemma 71

The number of saturating pushes performed is at most $\mathcal{O}(mn)$.

Proof.

- Suppose that we just made a saturating push along (u, v).

Lemma 71

The number of saturating pushes performed is at most $\mathcal{O}(mn)$.

Proof.

- Suppose that we just made a saturating push along (u, v).
- Hence, the edge (u, v) is deleted from the residual graph.

Lemma 71

The number of saturating pushes performed is at most $\mathcal{O}(mn)$.

Proof.

- Suppose that we just made a saturating push along (u, v).
- Hence, the edge (u, v) is deleted from the residual graph.
- For the edge to appear again, a push from v to u is required.

Lemma 71

The number of saturating pushes performed is at most $\mathcal{O}(mn)$.

Proof.

- Suppose that we just made a saturating push along (u, v).
- Hence, the edge (u, v) is deleted from the residual graph.
- For the edge to appear again, a push from v to u is required.
- Currently, $\ell(u) = \ell(v) + 1$, as we only make pushes along admissable edges.

Lemma 71

The number of saturating pushes performed is at most $\mathcal{O}(mn)$.

Proof.

- Suppose that we just made a saturating push along (u, v).
- Hence, the edge (u, v) is deleted from the residual graph.
- For the edge to appear again, a push from v to u is required.
- Currently, $\ell(u) = \ell(v) + 1$, as we only make pushes along admissable edges.
- For a push from v to u the edge (v, u) must become admissable. The label of v must increase by at least 2.

Lemma 71

The number of saturating pushes performed is at most $\mathcal{O}(mn)$.

Proof.

- Suppose that we just made a saturating push along (u, v).
- Hence, the edge (u, v) is deleted from the residual graph.
- For the edge to appear again, a push from v to u is required.
- Currently, $\ell(u) = \ell(v) + 1$, as we only make pushes along admissable edges.
- For a push from v to u the edge (v, u) must become admissable. The label of v must increase by at least 2.
- Since the label of v is at most $2n - 1$, there are at most n pushes along (u, v).

Lemma 72

The number of non-saturating pushes performed is at most $\mathcal{O}(n^2 m)$.

Lemma 72

The number of non-saturating pushes performed is at most $\mathcal{O}(n^2m)$.

Proof.

- Define a potential function $\Phi(f) = \sum_{\text{active nodes}v} \ell(v)$

Lemma 72

The number of non-saturating pushes performed is at most $\mathcal{O}(n^2 m)$.

Proof.

- Define a potential function $\Phi(f) = \sum_{\text{active nodes} v} \ell(v)$
- A saturating push increases Φ by at most $2n$.

Lemma 72

The number of non-saturating pushes performed is at most $\mathcal{O}(n^2 m)$.

Proof.

- Define a potential function $\Phi(f) = \sum_{\text{active nodes} v} \ell(v)$
- A saturating push increases Φ by at most $2n$.
- A relabel increases Φ by at most 1.

Lemma 72

The number of non-saturating pushes performed is at most $\mathcal{O}(n^2m)$.

Proof.

- Define a potential function $\Phi(f) = \sum_{\text{active nodes}v} \ell(v)$
- A saturating push increases Φ by at most $2n$.
- A relabel increases Φ by at most 1.
- A non-saturating push decreases Φ by at least 1 as the node that is pushed from becomes inactive and has a label that is strictly larger than the target.

Lemma 72

The number of non-saturating pushes performed is at most $\mathcal{O}(n^2m)$.

Proof.

- Define a potential function $\Phi(f) = \sum_{\text{active nodes}v} \ell(v)$
- A saturating push increases Φ by at most $2n$.
- A relabel increases Φ by at most 1.
- A non-saturating push decreases Φ by at least 1 as the node that is pushed from becomes inactive and has a label that is strictly larger than the target.
- Hence,

$$\begin{aligned}\#\text{non-saturating\_pushes} &\leq \#\text{relabels} + 2n \cdot \#\text{saturating\_pushes} \\ &\leq \mathcal{O}(n^2m) \ .\end{aligned}$$

Analysis

There is an implementation of the generic push relabel algorithm with running time $\mathcal{O}(n^2 m)$.

For every node maintain a list of admissable edges starting at that node. Further maintain a list of active nodes.

A push along an edge (u, v) can be performed in constant time

A relabel at a node u can be performed in time $\mathcal{O}(n)$

Analysis

There is an implementation of the generic push relabel algorithm with running time $\mathcal{O}(n^2 m)$.

For every node maintain a list of admissable edges starting at that node. Further maintain a list of active nodes.

A push along an edge (u, v) can be performed in constant time

A relabel at a node u can be performed in time $\mathcal{O}(n)$

Analysis

There is an implementation of the generic push relabel algorithm with running time $\mathcal{O}(n^2 m)$.

For every node maintain a list of admissable edges starting at that node. Further maintain a list of active nodes.

A push along an edge (u, v) can be performed in constant time

- check whether edge (v, u) needs to be added to G_f
- check whether (u, v) needs to be deleted (saturating push)
- check whether u becomes inactive and has to be deleted from the set of active nodes

A relabel at a node u can be performed in time $\mathcal{O}(n)$

Analysis

There is an implementation of the generic push relabel algorithm with running time $\mathcal{O}(n^2 m)$.

For every node maintain a list of admissable edges starting at that node. Further maintain a list of active nodes.

A push along an edge (u, v) can be performed in constant time

- check whether edge (v, u) needs to be added to G_f
- check whether (u, v) needs to be deleted (saturating push)
- check whether u becomes inactive and has to be deleted from the set of active nodes

A relabel at a node u can be performed in time $\mathcal{O}(n)$

Analysis

There is an implementation of the generic push relabel algorithm with running time $\mathcal{O}(n^2m)$.

For every node maintain a list of admissable edges starting at that node. Further maintain a list of active nodes.

A push along an edge (u,v) can be performed in constant time

- check whether edge (v,u) needs to be added to G_f
- check whether (u,v) needs to be deleted (saturating push)
- check whether u becomes inactive and has to be deleted from the set of active nodes

A relabel at a node u can be performed in time $\mathcal{O}(n)$

Analysis

There is an implementation of the generic push relabel algorithm with running time $\mathcal{O}(n^2m)$.

For every node maintain a list of admissable edges starting at that node. Further maintain a list of active nodes.

A push along an edge (u,v) can be performed in constant time

- check whether edge (v,u) needs to be added to G_f
- check whether (u,v) needs to be deleted (saturating push)
- check whether u becomes inactive and has to be deleted from the set of active nodes

A relabel at a node u can be performed in time $\mathcal{O}(n)$

- check for all outgoing edges if they become admissable
- check for all incoming edges if they become non-admissable

Analysis

There is an implementation of the generic push relabel algorithm with running time $\mathcal{O}(n^2m)$.

For every node maintain a list of admissable edges starting at that node. Further maintain a list of active nodes.

A push along an edge (u,v) can be performed in constant time

- check whether edge (v,u) needs to be added to G_f
- check whether (u,v) needs to be deleted (saturating push)
- check whether u becomes inactive and has to be deleted from the set of active nodes

A relabel at a node u can be performed in time $\mathcal{O}(n)$

- check for all outgoing edges if they become admissable
- check for all incoming edges if they become non-admissable

13.2 Relabel to front

For special variants of push relabel algorithms we organize the neighbours of a node into a linked list (possible neighbours in the residual graph G_f). Then we use the discharge-operation:

Algorithm 48 discharge(u)

```
1: while u is active do
2:     v ← u.current-neighbour
3:     if v = null then
4:         relabel(u)
5:         u.current-neighbour ← u.neighbour-list-head
6:     else
7:         if (u, v) admissable then push(u, v)
8:         else u.current-neighbour ← v.next-in-list
```

13.2 Relabel to front

Lemma 73

If $v = \text{null}$ in line 3, then there is no outgoing admissable edge from u.

The lemma holds because push- and relabel-operations on nodes different from u cannot make edges outgoing from u admissable.

This shows that discharge(u) is correct, and that we can perform a relabel in line 4.

13.2 Relabel to front

Algorithm 49 relabel-to-front(G, s, t)

```
 1: initialize preflow
 2: initialize node list L containing V \ {s, t} in any order
 3: foreach u ∈ V \ {s, t} do
 4:     u.current-neighbour ← u.neighbour-list-head
 5: u ← L.head
 6: while u ≠ null do
 7:     old-height ← ℓ(u)
 8:     discharge(u)
 9:     if ℓ(u) > old-height then
10:         move u to the front of L
11:     u ← u.next
```

13.2 Relabel to front

Lemma 74 (Invariant)

In Line 6 of the relabel-to-front algorithm the following invariant holds.

1. *The sequence L is topologically sorted w.r.t. the set of admissable edges; this means for an admissable edge (x, y) the node x appears before y in sequence L.*
2. *No node before u in the list L is active.*

Proof:

- Initialization:
  1. In the beginning s has label $n \geq 2$, and all other nodes have label 0. Hence, no edge is admissable, which means that any ordering L is permitted.
  2. We start with u being the head of the list; hence no node before u can be active
- Maintenance:
  1.
     - Pushes do no create any new admissable edges. Therefore, not relabeling u leaves L topologically sorted.
     - After relabeling, u cannot have admissable incoming edges as such an edge (x, u) would have had a difference $\ell(x) - \ell(u) \geq 2$ before the re-labeling (such edges do not exist in the residual graph).
       Hence, moving u to the front does not violate the sorting property for any edge; however it fixes this property for all admissable edges leaving u that were generated by the relabeling.

13.2 Relabel to front

Proof:

- Maintenance:
  2. If we do a relabel there is nothing to prove because the only node before u' (u in the next iteration) will be the current u; the discharge(u) operation only terminates when u is not active anymore.

     For the case that we do a relabel, observe that the only way a predecessor could be active is that we push flow to it via an admissable arc. However, all admissable arc point to successors of u.

Note that the invariant for $u = \text{null}$ means that we have a preflow with a valid labelling that does not have active nodes. This means we have a maximum flow.

13.2 Relabel to front

Lemma 75

There are at most $\mathcal{O}(n^3)$ calls to discharge(u).

Every discharge operation without a relabel advances u (the current node within list L). Hence, if we have n discharge operations without a relabel we have $u = \text{null}$ and the algorithm terminates.

Therefore, the number of calls to discharge is at most $n(\#relabels + 1) = \mathcal{O}(n^3)$.

13.2 Relabel to front

Lemma 76

The cost for all relabel-operations is only $\mathcal{O}(n^2)$.

A relabel-operation at a node is constant time (increasing the label and resetting $u.\mathit{current\text{-}neighbour}$). In total we have $\mathcal{O}(n^2)$ relabel-operations.

13.2 Relabel to front

Note that by definition a saturing push operation ($\min\{c_f(e), f(u)\} = c_f(e)$) can at the same time be a non-saturating push operation ($\min\{c_f(e), f(u)\} = f(u)$).

Lemma 77

*The cost for all saturating push-operations that are **not** also non-saturating push-operations is only $\mathcal{O}(mn)$.*

Note that such a push-operation leaves the node u active but makes the edge e disappear from the residual graph. Therefore the push-operation is immediately followed by an increase of the pointer $u.\mathit{current\text{-}neighbour}$.
This pointer can traverse the neighbour-list at most $\mathcal{O}(n)$ times (upper bound on number of relabels) and the neighbour-list has only $\mathit{degree}(u) + 1$ many entries ($+1$ for null-entry).

13.2 Relabel to front

Lemma 78

The cost for all non-saturating push-operations is only $\mathcal{O}(n^3)$.

A non-saturating push-operation takes constant time and ends the current call to discharge(). Hence, there are only $\mathcal{O}(n^3)$ such operations.

Theorem 79

The push-relabel algorithm with the rule relabel-to-front takes time $\mathcal{O}(n^3)$.

13.3 Highest label

Algorithm 50 highest-label(G, s, t)

```
1: initialize preflow
2: foreach u ∈ V \ {s, t} do
3:     u.current-neighbour ← u.neighbour-list-head
4: while ∃ active node u do
5:     select active node u with highest label
6:     discharge(u)
```

13.3 Highest label

Lemma 80

When using highest label the number of non-saturating pushes is only $\mathcal{O}(n^3)$.

After a non-saturating push from u a relabel is required to make a currently non-active node x, with $\ell(x) \geq \ell(u)$ active again (note that this includes u).

Hence, after n non-saturating pushes without an intermediate relabel there are no active nodes left.

Therefore, the number of non-saturating pushes is at most $n(\#relabels + 1) = \mathcal{O}(n^3)$.

13.3 Highest label

Since a discharge-operation is terminated by a non-saturating push this gives an upper bound of $\mathcal{O}(n^3)$ on the number of discharge-operations.

The cost for relabels and saturating pushes can be estimated in exactly the same way as in the case of relabel-to-front.

Question:
How do we find the next node for a discharge operation?

13.3 Highest label

Maintain lists L_i, $i \in \{0, \ldots, 2n\}$, where list L_i contains active nodes with label i (maintaining these lists induces only constant additional cost for every push-operation and for every relabel-operation).

After a discharge operation terminated for a node u with label k, traverse the lists $k-1, \ldots, 0$, (in that order) until you find a non-empty list.

Unless the last (non-saturating) push was to s or t the list $k-1$ must be non-empty (i.e., the search takes constant time).

13.3 Highest label

Hence, the total time required for searching for active nodes is at most

$$\mathcal{O}(n^3) + n(\#\textit{non-saturating-pushes-to-s-or-t})$$

Lemma 81

The number of non-saturating pushes to s or t is at most $\mathcal{O}(n^2)$.

With this lemma we get

Theorem 82

The push-relabel algorithm with the rule highest-label takes time $\mathcal{O}(n^3)$.

13.3 Highest label

Proof of the Lemma.

- We only show that the number of pushes to the source is at most $\mathcal{O}(n^2)$. A similar argument holds for the target.
- After a node v (which must have $\ell(v) = n+1$) made a non-saturating push to the source there needs to be another node whose label is increased from $\leq n+1$ to $n+2$ before v can become active again.
- This happens for every push that v makes to the source. Since, every node can pass the threshold $n+2$ at most once, v can make at most n pushes to the source.
- As this holds for every node the total number of pushes to the source is at most $\mathcal{O}(n^2)$.