# CMPSC 465
# Data Structures and Algorithms
# Spring 2022

Instructor: Chunhao Wang

# Flow network
# (Textbook, Section 7.2
# Kleinberg & Tardos Section 7.1)

## Motivation

We use graphs to model "transportation networks"

# Motivation

We use graphs to model "transportation networks"
i.e., networks whose edges allow some sort of traffic, and vertices act as switches

# Motivation

We use graphs to model "transportation networks"
i.e., networks whose edges allow some sort of traffic, and vertices act as switches

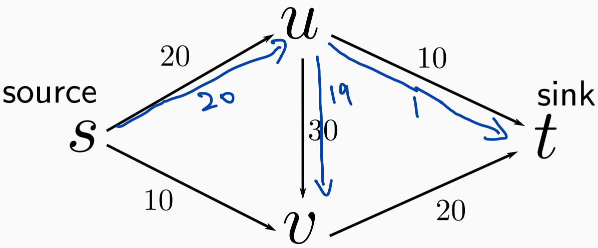
In this setting, edges have **capacities**, and we assume there is a **source vertex** generating traffic, a **sink vertex**, which absorbs traffic.

## Motivation

We use graphs to model "transportation networks"
i.e., networks whose edges allow some sort of traffic, and vertices act as switches

In this setting, edges have **capacities**, and we assume there is a **source vertex** generating traffic, a **sink vertex**, which absorbs traffic.

Example:



# Flow network

**Definition**

A **flow network** is a directed graph $G = (V, E)$ s.t.

# Flow network

**Definition**

A **flow network** is a directed graph $G = (V, E)$ s.t.

1. there is a capacity $c_e \geq 0$ for all $e \in E$

# Flow network

## Definition

A **flow network** is a directed graph $G = (V, E)$ s.t.

1. there is a capacity $c_e \geq 0$ for all $e \in E$
2. there is a single source $s \in V$

# Flow network

**Definition**

A **flow network** is a directed graph $G = (V, E)$ s.t.

1. there is a capacity $c_e \geq 0$ for all $e \in E$
2. there is a single source $s \in V$
3. there is a single sink $t \in V$

## $s$-$t$ flow

### Definition

An **s-t flow** is a function $f : E \to \mathbb{R}^{\geq 0}$ satisfying

# $s$-$t$ flow

## Definition

An **s-t flow** is a function $f : E \to \mathbb{R}^{\geq 0}$ satisfying

- **Capacity constraint:** $\forall e \in E$, $0 \leq f(e) \leq c_e$

# $s$-$t$ flow

## Definition

An **s-t flow** is a function $f : E \to \mathbb{R}^{\geq 0}$ satisfying

- **Capacity constraint:** $\forall e \in E$, $0 \leq f(e) \leq c_e$
- **Conservation condition:** $\forall v \in V - \{s, t\}$

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

# $s$-$t$ flow

## Definition

An **s-t flow** is a function $f : E \to \mathbb{R}^{\geq 0}$ satisfying

- **Capacity constraint:** $\forall e \in E$, $0 \leq f(e) \leq c_e$
- **Conservation condition:** $\forall v \in V - \{s, t\}$

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Intuition: $f(e)$ is the amount of flow/traffic carried out by $e$

**Definition**

An **s-t flow** is a function $f : E \to \mathbb{R}^{\geq 0}$ satisfying

- **Capacity constraint:** $\forall e \in E$, $0 \leq f(e) \leq c_e$
- **Conservation condition:** $\forall v \in V - \{s, t\}$

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Intuition: $f(e)$ is the amount of flow/traffic carried out by $e$

**Definition**

The **value** of a flow $f$ is

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

## $s$-$t$ flow

**Definition**

An **s-t flow** is a function $f : E \to \mathbb{R}^{\geq 0}$ satisfying

- **Capacity constraint:** $\forall e \in E$, $0 \leq f(e) \leq c_e$
- **Conservation condition:** $\forall v \in V - \{s, t\}$

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Intuition: $f(e)$ is the amount of flow/traffic carried out by $e$

**Definition**

The **value** of a flow $f$ is

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

Intuition: $v(f)$ shows how much traffic can be accommodated

# The max-flow problem

**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

# The max-flow problem

**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

Natural idea: Find an $s$-$t$ path and push a flow of value $\min\{c_e : e \text{ along the path}\}$. Then keep doing this

# The max-flow problem

**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

Natural idea: Find an $s$-$t$ path and push a flow of value $\min\{c_e : e \text{ along the path}\}$. Then keep doing this

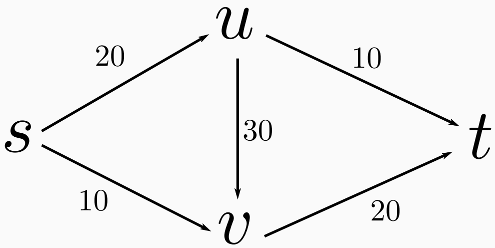
Example:

**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

Natural idea: Find an $s$-$t$ path and push a flow of value $\min\{c_e : e \text{ along the path}\}$. Then keep doing this

Example:

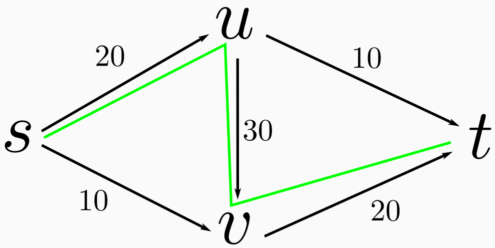


**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

Natural idea: Find an $s$-$t$ path and push a flow of value $\min\{c_e : e \text{ along the path}\}$. Then keep doing this

Example:



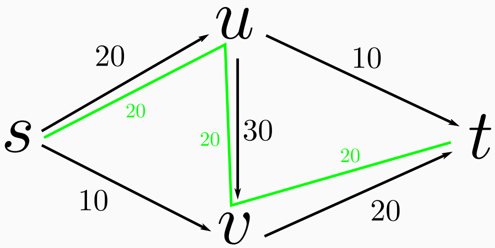
# The max-flow problem

**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

Natural idea: Find an $s$-$t$ path and push a flow of value $\min\{c_e : e \text{ along the path}\}$. Then keep doing this

Example:

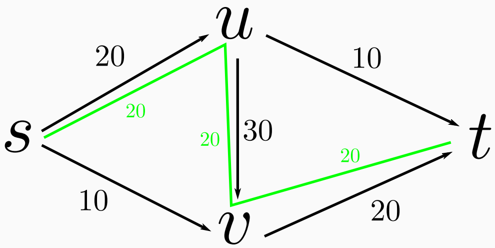


**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

Natural idea: Find an $s$-$t$ path and push a flow of value $\min\{c_e : e \text{ along the path}\}$. Then keep doing this

Example:




$V(f)=20$

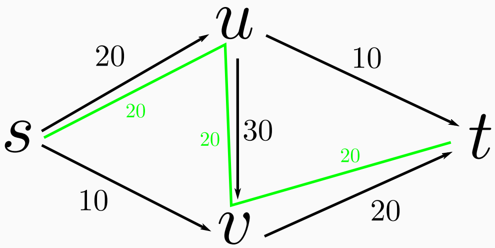
$$f(s,u) = 20$$
$$f(u,v) = 20$$
$$f(v,t) = 20$$
$$f(s,v) = 0$$
$$f(u,t) = 0$$

**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

Natural idea: Find an $s$-$t$ path and push a flow of value $\min\{c_e : e \text{ along the path}\}$. Then keep doing this

Example:




$$f(s,u) = 20$$
$$f(u,v) = 20$$
$$f(v,t) = 20$$
$$f(s,v) = 0$$
$$f(u,t) = 0$$

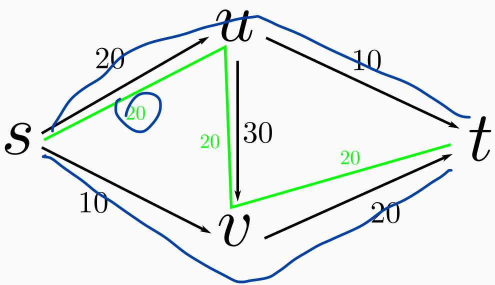
$v(f) = 20$.

**Problem (The Max-Flow problem)**

*Given a flow network, find a flow of the maximum possible value*

Natural idea: Find an $s$-$t$ path and push a flow of value $\min\{c_e : e \text{ along the path}\}$. Then keep doing this

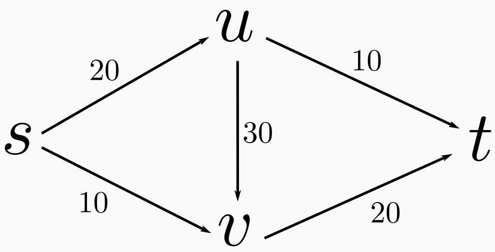
Example:



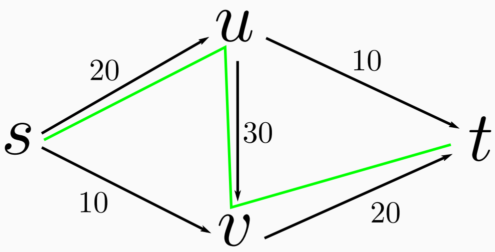

$$f(s,u) = 20$$
$$f(u,v) = 20$$
$$f(v,t) = 20$$
$$f(s,v) = 0$$
$$f(u,t) = 0$$

$v(f) = 20$. Can we do better?

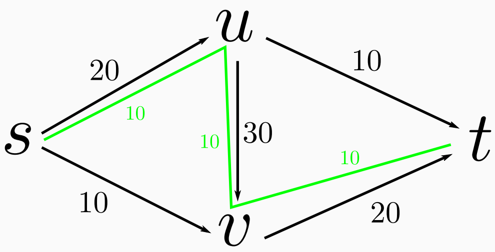
# A better flow



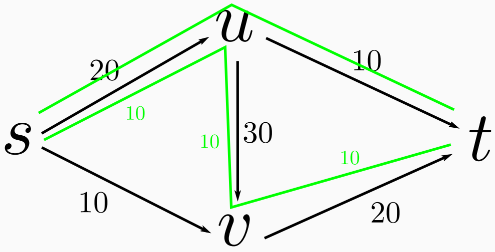
# A better flow



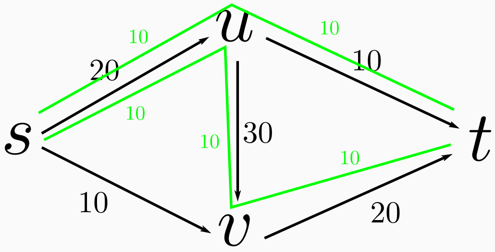
# A better flow



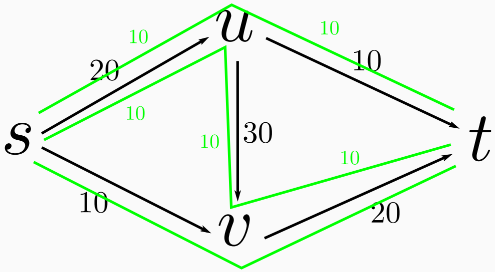
# A better flow



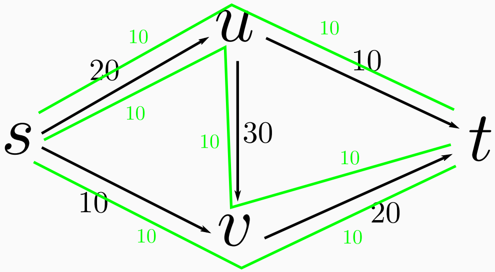
# A better flow



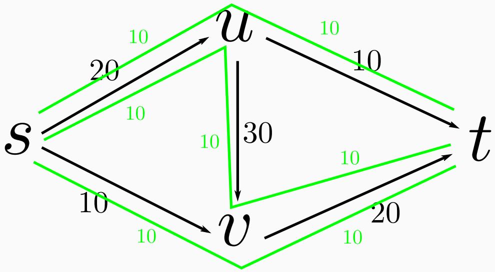
# A better flow



# A better flow



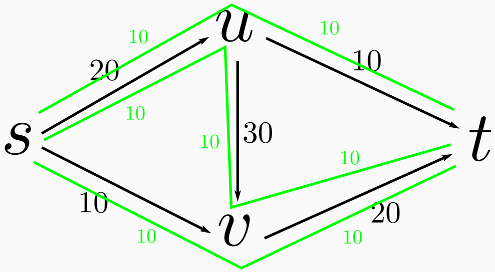
# A better flow




$$f(s,u) = 20$$
$$f(u,v) = 10$$
$$f(v,t) = 20$$
$$f(s,v) = 10$$
$$f(u,t) = 10$$

$v(f) =$

# A better flow




$$f(s,u) = 20$$
$$f(u,v) = 10$$
$$f(v,t) = 20$$
$$f(s,v) = 10$$
$$f(u,t) = 10$$

$v(f) = 30.$

# Residual graph

We use a residual graph to keep track of the "leftover" capacity

# Residual graph

We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

# Residual graph

We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

- the vertex set of $G_f$ is the same as $G$

# Residual graph

We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

- the vertex set of $G_f$ is the same as $G$
- for each edge $e$ of $G$ with $f(e) < c_e$ (leftover),

# Residual graph

We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

- the vertex set of $G_f$ is the same as $G$
- for each edge $e$ of $G$ with $f(e) < c_e$ (leftover), include $e$ in $G_f$ with capacity $c_e - f(e)$ (**forward edge**)

# Residual graph

We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

- the vertex set of $G_f$ is the same as $G$
- for each edge $e$ of $G$ with $f(e) < c_e$ (leftover), include $e$ in $G_f$ with capacity $c_e - f(e)$ (**forward edge**)
- for each edge $e = (u, v)$ in $G$ with $f(e) > 0$ (how much we can undo),

## Residual graph

We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

- the vertex set of $G_f$ is the same as $G$
- for each edge $e$ of $G$ with $f(e) < c_e$ (leftover), include $e$ in $G_f$ with capacity $c_e - f(e)$ (**forward edge**)
- for each edge $e = (u, v)$ in $G$ with $f(e) > 0$ (how much we can undo), include $e' = (v, u)$ in $G_f$ with capacity $f(e)$ (**backward edge**)

# Residual graph

We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

- the vertex set of $G_f$ is the same as $G$
- for each edge $e$ of $G$ with $f(e) < c_e$ (leftover), include $e$ in $G_f$ with capacity $c_e - f(e)$ (**forward edge**)
- for each edge $e = (u, v)$ in $G$ with $f(e) > 0$ (how much we can undo), include $e' = (v, u)$ in $G_f$ with capacity $f(e)$ (**backward edge**)

Capacity of an edge in the residual graph is called **residual capacity**

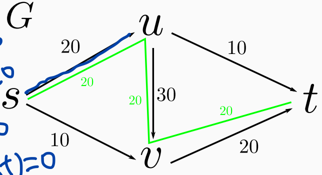
We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

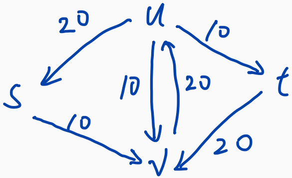
- the vertex set of $G_f$ is the same as $G$
- for each edge $e$ of $G$ with $f(e) < c_e$ (leftover), include $e$ in $G_f$ with capacity $\boxed{c_e - f(e)}$ (**forward edge**)
- for each edge $e = (u, v)$ in $G$ with $f(e) > 0$ (how much we can undo), include $e' = (v, u)$ in $G_f$ with capacity $f(e)$ (**backward edge**)

Capacity of an edge in the residual graph is called **residual capacity**

$f(s,u)=20$
$f(u,v)=20$
$f(v,t)=20$
$f(s,v)=0$
$f(u,t)=0$




$G_f$:



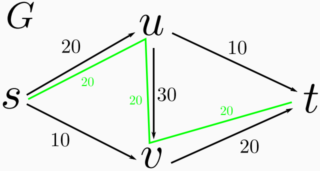
# Residual graph

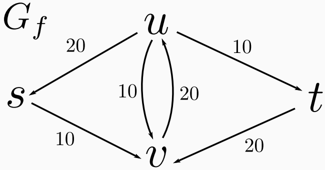
We use a residual graph to keep track of the "leftover" capacity

Given a flow network $G$, flow $f$, construct the **residual graph** $G_f$ as follows

- the vertex set of $G_f$ is the same as $G$
- for each edge $e$ of $G$ with $f(e) < c_e$ (leftover), include $e$ in $G_f$ with capacity $c_e - f(e)$ (**forward edge**)
- for each edge $e = (u, v)$ in $G$ with $f(e) > 0$ (how much we can undo), include $e' = (v, u)$ in $G_f$ with capacity $f(e)$ (**backward edge**)

Capacity of an edge in the residual graph is called **residual capacity**



# Better idea

We introduce **augmenting paths**

# Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$,

# Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

# Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\text{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

## Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\mathrm{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

**def** $\textsc{Augment}(f, P)$**:**

## Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\mathrm{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

**def** $\textsc{Augment}(f, P)$**:**

  Set $b = \mathrm{bottleneck}(P, f)$;

# Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\text{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

**def** $\textsc{Augment}(f, P)$**:**

    Set $b = \text{bottleneck}(P, f)$;

    **for** *each* $e = (u, v) \in P$**:**

## Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\mathrm{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

```
def AUGMENT(f, P):
    Set b = bottleneck(P, f);
    for each e = (u, v) ∈ P:
        if e is a forward edge:
```

# Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\mathrm{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

```
def AUGMENT(f, P):
    Set b = bottleneck(P, f);
    for each e = (u, v) ∈ P:
        if e is a forward edge:
            Increase f(e) by b;
```

# Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\mathrm{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

**def** $\textsc{Augment}(f, P)$**:**

  Set $b = \mathrm{bottleneck}(P, f)$;

  **for** *each* $e = (u, v) \in P$**:**

    **if** *$e$ is a forward edge***:**

      Increase $f(e)$ by $b$;

    **else:**

## Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\text{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

```
def AUGMENT(f, P):
    Set b = bottleneck(P, f);
    for each e = (u, v) ∈ P:
        if e is a forward edge:
            Increase f(e) by b;
        else:
            Decrease f(e) by b;
```

# Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\mathrm{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

**def** $\textsc{Augment}(f, P)$**:**

- Set $b = \mathrm{bottleneck}(P, f)$;
- **for** *each* $e = (u, v) \in P$**:**
  - **if** *e is a forward edge***:**
    - Increase $f(e)$ by $b$;
  - **else:**
    - Decrease $f(e)$ by $b$;
- **return** $f$;

# Better idea

We introduce **augmenting paths**

Let $P$ be an $s$-$t$ path in $G_f$, we can improve our flow by augmenting along $P$ as follows

Define $\mathrm{bottleneck}(P, f)$ as the minimum residual capacity of edges of $P$

**def** $\textsc{Augment}(f, P)$**:**

```
Set b = bottleneck(P, f);
for each e = (u, v) ∈ P:
    if e is a forward edge:
        Increase f(e) by b;
    else:
        Decrease f(e) by b;
return f;
```

Any $s$-$t$ path in $G_f$ is called an **augmenting path**

# The Ford-Fulkerson algorithm

**def** FORD-FULKERSON(*G, capacities c*):

# The Ford-Fulkerson algorithm

**def** FORD-FULKERSON(*G, capacities c*)**:**

  Set $f(e) = 0$ for all $e \in G$;

# The Ford-Fulkerson algorithm

**def** FORD-FULKERSON*(G, capacities c)***:**

Set $f(e) = 0$ for all $e \in G$;

**while** $\exists$ *s-t path in the residual graph* $G_f$**:**

# The Ford-Fulkerson algorithm

**def** FORD-FULKERSON(*G, capacities c*)**:**

Set $f(e) = 0$ for all $e \in G$;

**while** $\exists$ *s-t path in the residual graph* $G_f$**:**

Let $P$ be a simple $s$-$t$ path in $G_f$;

# The Ford-Fulkerson algorithm

**def** FORD-FULKERSON(*G, capacities c*)**:**

  Set $f(e) = 0$ for all $e \in G$;

  **while** $\exists$ *s-t path in the residual graph* $G_f$**:**

    Let $P$ be a simple $s$-$t$ path in $G_f$;

    $f' = \textsc{Augment}(f, P)$;

# The Ford-Fulkerson algorithm

**def** $\textsc{Ford-Fulkerson}$*(G, capacities c)*:

Set $f(e) = 0$ for all $e \in G$;

**while** $\exists$ *s-t path in the residual graph* $G_f$:

Let $P$ be a simple $s$-$t$ path in $G_f$;

$f' = \textsc{Augment}(f, P)$;

$G_f = G_{f'}$;

# The Ford-Fulkerson algorithm

**def** $\text{FORD-FULKERSON}$*(G, capacities c)*:

Set $f(e) = 0$ for all $e \in G$;

**while** $\exists$ *s-t path in the residual graph* $G_f$:

Let $P$ be a simple $s$-$t$ path in $G_f$;

$f' = \text{AUGMENT}(f, P)$;

$G_f = G_{f'}$;

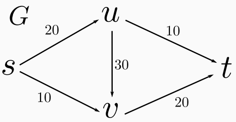
$f = f'$;

# The Ford-Fulkerson algorithm

**def** $\textsc{Ford-Fulkerson}$*(G, capacities c)***:**

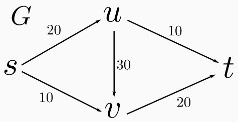
- Set $f(e) = 0$ for all $e \in G$;
- **while** $\exists$ *s-t path in the residual graph* $G_f$**:**
  - Let $P$ be a simple $s$-$t$ path in $G_f$;
  - $f' = \textsc{Augment}(f, P)$;
  - $G_f = G_{f'}$;
  - $f = f'$;
- **return** $f$;

# Running example




| $e$ | f(e) | | |
|---|---|---|---|
| $s \to u$ | 0 | | |
| $s \to v$ | 0 | | |
| $u \to v$ | 0 | | |
| $u \to t$ | 0 | | |
| $v \to t$ | 0 | | |

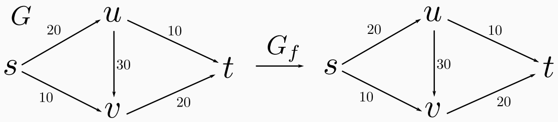
# Running example




$G_f$:

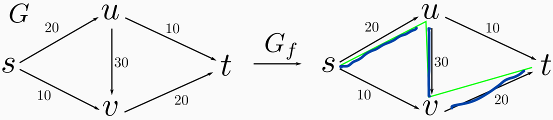
| $e$ | $f_1(e)$ | | |
|---|---|---|---|
| $s \to u$ | 0 | | |
| $s \to v$ | 0 | | |
| $u \to v$ | 0 | | |
| $u \to t$ | 0 | | |
| $v \to t$ | 0 | | |

# Running example




| $e$ | $f_1(e)$ | | |
|---|---|---|---|
| $s \to u$ | 0 | | |
| $s \to v$ | 0 | | |
| $u \to v$ | 0 | | |
| $u \to t$ | 0 | | |
| $v \to t$ | 0 | | |

# Running example

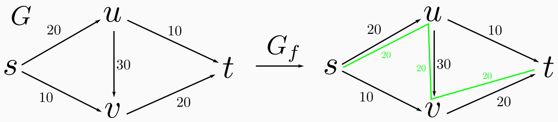



$P : s \to u \to v \to t$

b: 20

| $e$ | $f_1(e)$ | | |
|---|---|---|---|
| $s \to u$ | 0 | 20 | |
| $s \to v$ | 0 | 0 | |
| $u \to v$ | 0 | 20 | |
| $u \to t$ | 0 | 0 | |
| $v \to t$ | 0 | 20 | |

# Running example

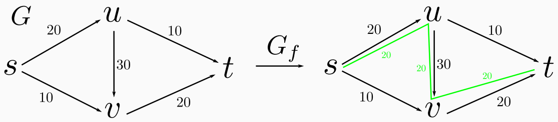



$P: s \to u \to v \to t$

$b: 20$

| $e$ | $f_1(e)$ | | |
|---|---|---|---|
| $s \to u$ | 0 | | |
| $s \to v$ | 0 | | |
| $u \to v$ | 0 | | |
| $u \to t$ | 0 | | |
| $v \to t$ | 0 | | |

# Running example

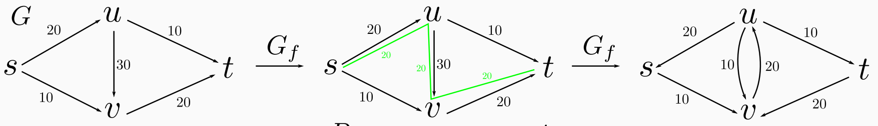



$P : s \to u \to v \to t$

$b : 20$

| $e$ | $f_1(e)$ | $f_2(e)$ | |
|---|---|---|---|
| $s \to u$ | 0 | 20 | |
| $s \to v$ | 0 | 0 | |
| $u \to v$ | 0 | 20 | |
| $u \to t$ | 0 | 0 | |
| $v \to t$ | 0 | 20 | |

# Running example

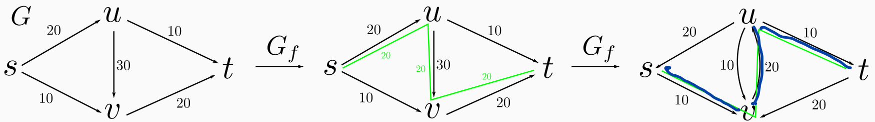



$P : s \to u \to v \to t$

$b : 20$

| $e$ | $f_1(e)$ | $f_2(e)$ | |
|---|---|---|---|
| $s \to u$ | 0 | 20 | |
| $s \to v$ | 0 | 0 | |
| $u \to v$ | 0 | 20 | |
| $u \to t$ | 0 | 0 | |
| $v \to t$ | 0 | 20 | |

# Running example




$P : s \to u \to v \to t$

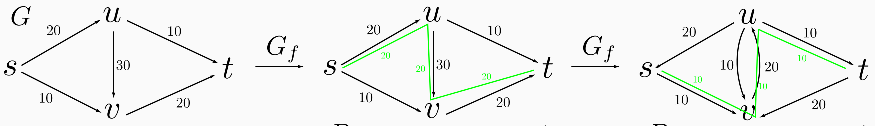
$b : 20$

$P : s \to v \to u \to t$

$b : 10$

| $e$ | $f_1(e)$ | $f_2(e)$ | |
|---|---|---|---|
| $s \to u$ | 0 | 20 | 20 |
| $s \to v$ | 0 | 0 | 10 |
| $u \to v$ | 0 | 20 | 10 |
| $u \to t$ | 0 | 0 | 10 |
| $v \to t$ | 0 | 20 | 20 |

# Running example

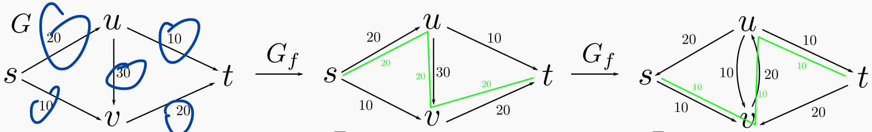



$P : s \to u \to v \to t$
$b : 20$

$P : s \to v \to u \to t$
$b : 10$

| $e$ | $f_1(e)$ | $f_2(e)$ | |
|---|---|---|---|
| $s \to u$ | 0 | 20 | |
| $s \to v$ | 0 | 0 | |
| $u \to v$ | 0 | 20 | |
| $u \to t$ | 0 | 0 | |
| $v \to t$ | 0 | 20 | |

# Running example

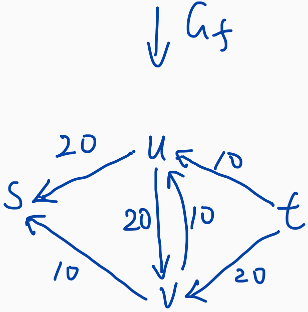



$P: s \to u \to v \to t$

$b: 20$

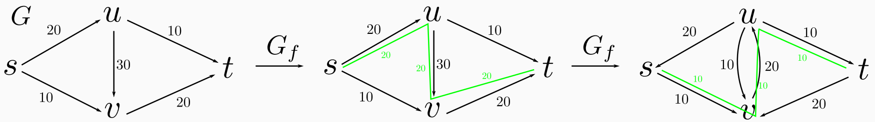
$P: s \to v \to u \to t$

$b: 10$




| $e$ | $f_1(e)$ | $f_2(e)$ | $f_3(e)$ |
|---|---|---|---|
| $s \to u$ | 0 | 20 | 20 |
| $s \to v$ | 0 | 0 | 10 |
| $u \to v$ | 0 | 20 | 10 |
| $u \to t$ | 0 | 0 | 10 |
| $v \to t$ | 0 | 20 | 20 |

# Running example

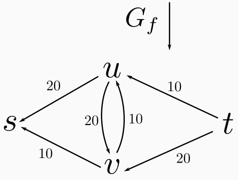



$P: s \to u \to v \to t$

$b: 20$

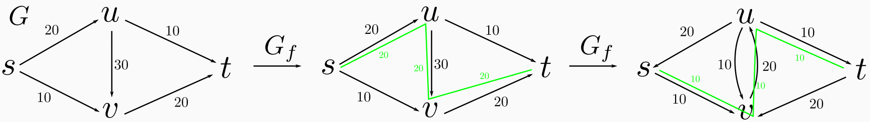
$P: s \to v \to u \to t$

$b: 10$




| $e$ | $f_1(e)$ | $f_2(e)$ | $f_3(e)$ |
|---|---|---|---|
| $s \to u$ | 0 | 20 | 20 |
| $s \to v$ | 0 | 0 | 10 |
| $u \to v$ | 0 | 20 | 10 |
| $u \to t$ | 0 | 0 | 10 |
| $v \to t$ | 0 | 20 | 20 |

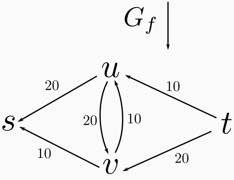
# Running example




$P: s \to u \to v \to t$
$b: 20$

$P: s \to v \to u \to t$
$b: 10$

| $e$ | $f_1(e)$ | $f_2(e)$ | $f_3(e)$ |
|---|---|---|---|
| $s \to u$ | 0 | 20 | 20 |
| $s \to v$ | 0 | 0 | 10 |
| $u \to v$ | 0 | 20 | 10 |
| $u \to t$ | 0 | 0 | 10 |
| $v \to t$ | 0 | 20 | 20 |




No more $s$-$t$ path