# Network Flow Algorithms

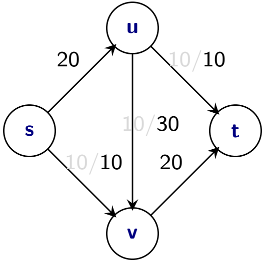
Lecture 17

March 29, 2011

Part I

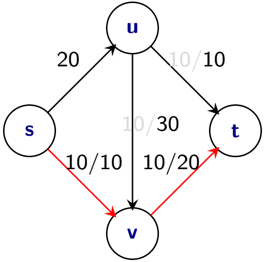
# Algorithm(s) for Maximum Flow

# Greedy Approach



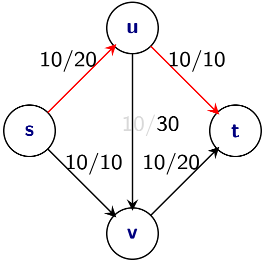

1. Begin with $f(e) = 0$ for each edge
2. Find a $s$-$t$ path $P$ with $f(e) < c(e)$ for every edge $e \in P$
3. *Augment* flow along this path
4. Repeat augmentation for as long as possible.

# Greedy Approach



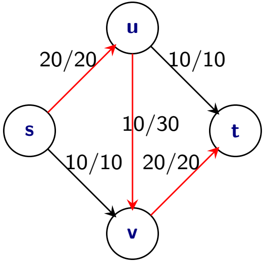

1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e \in P}$
3. *Augment* flow along this path
4. Repeat augmentation for as long as possible.

# Greedy Approach



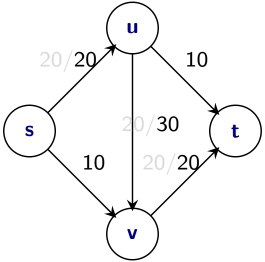

1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e \in P}$
3. *Augment* flow along this path
4. Repeat augmentation for as long as possible.

# Greedy Approach




1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e \in P}$
3. *Augment* flow along this path
4. Repeat augmentation for as long as possible.

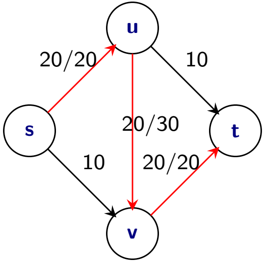
# Greedy Approach: Issues




1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!
Need to "push-back" flow along edge $\mathbf{(u, v)}$

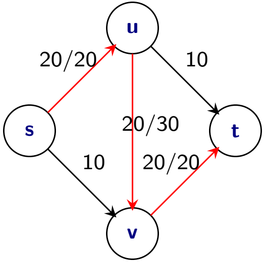
# Greedy Approach: Issues




1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!
Need to "push-back" flow along edge $\mathbf{(u, v)}$

# Greedy Approach: Issues

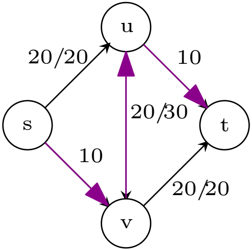



1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e \in P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!

Need to "push-back" flow along edge $\mathbf{(u, v)}$

# Greedy Approach: Issues




1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e \in P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!
Need to "push-back" flow along edge $\mathbf{(u, v)}$

# Residual Graph

## Definition

For a network $G = (V, E)$ and flow $f$, the residual graph $G_f = (V', E')$ of $G$ with respect to $f$ is

- $V' = V$
- Forward Edges: For each edge $e \in E$ with $f(e) < c(e)$, we $e \in E'$ with capacity $c(e) - f(e)$
- Backward Edges: For each edge $e = (u, v) \in E$ with $f(e) > 0$, we $(v, u) \in E'$ with capacity $f(e)$

# Residual Graph

## Definition

For a network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and flow $\mathbf{f}$, the residual graph $\mathbf{G_f} = (\mathbf{V'}, \mathbf{E'})$ of $\mathbf{G}$ with respect to $\mathbf{f}$ is

- $\mathbf{V'} = \mathbf{V}$
- Forward Edges: For each edge $\mathbf{e} \in \mathbf{E}$ with $\mathbf{f(e)} < \mathbf{c(e)}$, we $\mathbf{e} \in \mathbf{E'}$ with capacity $\mathbf{c(e)} - \mathbf{f(e)}$
- Backward Edges: For each edge $\mathbf{e} = (\mathbf{u}, \mathbf{v}) \in \mathbf{E}$ with $\mathbf{f(e)} > \mathbf{0}$, we $(\mathbf{v}, \mathbf{u}) \in \mathbf{E'}$ with capacity $\mathbf{f(e)}$

# Residual Graph

## Definition

For a network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and flow $\mathbf{f}$, the residual graph $\mathbf{G_f} = (\mathbf{V'}, \mathbf{E'})$ of $\mathbf{G}$ with respect to $\mathbf{f}$ is

- $\mathbf{V'} = \mathbf{V}$
- Forward Edges: For each edge $\mathbf{e} \in \mathbf{E}$ with $\mathbf{f(e)} < \mathbf{c(e)}$, we $\mathbf{e} \in \mathbf{E'}$ with capacity $\mathbf{c(e)} - \mathbf{f(e)}$
- Backward Edges: For each edge $\mathbf{e} = (\mathbf{u}, \mathbf{v}) \in \mathbf{E}$ with $\mathbf{f(e)} > \mathbf{0}$, we $(\mathbf{v}, \mathbf{u}) \in \mathbf{E'}$ with capacity $\mathbf{f(e)}$

# Residual Graph Example

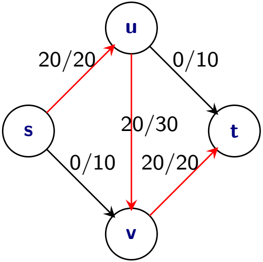



Figure: Flow in red edges

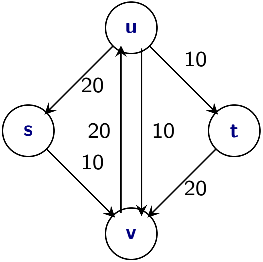



Figure: Residual Graph

# Residual Graph Property

**Observation:** Residual graph captures the "residual" problem exactly.

**Lemma**

*Let $\mathbf{f}$ be a flow in $\mathbf{G}$ and $\mathbf{G}_\mathbf{f}$ be the residual graph. If $\mathbf{f}'$ is a flow in $\mathbf{G}_\mathbf{f}$ then $\mathbf{f} + \mathbf{f}'$ is a flow in $\mathbf{G}$ of value $\mathbf{v}(\mathbf{f}) + \mathbf{v}(\mathbf{f}')$.*

**Lemma**

*Let $\mathbf{f}$ and $\mathbf{f}'$ be two flows in $\mathbf{G}$ with $\mathbf{v}(\mathbf{f}') \geq \mathbf{v}(\mathbf{f})$. Then there is a flow $\mathbf{f}''$ of value $\mathbf{v}(\mathbf{f}') - \mathbf{v}(\mathbf{f})$ in $\mathbf{G}_\mathbf{f}$.*

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

# Residual Graph Property

**Observation:** Residual graph captures the "residual" problem exactly.

**Lemma**

*Let $\mathbf{f}$ be a flow in $\mathbf{G}$ and $\mathbf{G_f}$ be the residual graph. If $\mathbf{f'}$ is a flow in $\mathbf{G_f}$ then $\mathbf{f} + \mathbf{f'}$ is a flow in $\mathbf{G}$ of value $\mathbf{v(f)} + \mathbf{v(f')}$.*

**Lemma**

*Let $\mathbf{f}$ and $\mathbf{f'}$ be two flows in $\mathbf{G}$ with $\mathbf{v(f')} \geq \mathbf{v(f)}$. Then there is a flow $\mathbf{f''}$ of value $\mathbf{v(f')} - \mathbf{v(f)}$ in $\mathbf{G_f}$.*

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

# Residual Graph Property

**Observation:** Residual graph captures the "residual" problem exactly.

**Lemma**

*Let* $\mathbf{f}$ *be a flow in* $\mathbf{G}$ *and* $\mathbf{G_f}$ *be the residual graph. If* $\mathbf{f'}$ *is a flow in* $\mathbf{G_f}$ *then* $\mathbf{f} + \mathbf{f'}$ *is a flow in* $\mathbf{G}$ *of value* $\mathbf{v(f)} + \mathbf{v(f')}$.

**Lemma**

*Let* $\mathbf{f}$ *and* $\mathbf{f'}$ *be two flows in* $\mathbf{G}$ *with* $\mathbf{v(f')} \geq \mathbf{v(f)}$. *Then there is a flow* $\mathbf{f''}$ *of value* $\mathbf{v(f')} - \mathbf{v(f)}$ *in* $\mathbf{G_f}$.

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

# Residual Graph Property

**Observation:** Residual graph captures the "residual" problem exactly.

## Lemma

*Let $\mathbf{f}$ be a flow in $\mathbf{G}$ and $\mathbf{G_f}$ be the residual graph. If $\mathbf{f'}$ is a flow in $\mathbf{G_f}$ then $\mathbf{f} + \mathbf{f'}$ is a flow in $\mathbf{G}$ of value $\mathbf{v(f)} + \mathbf{v(f')}$.*

## Lemma

*Let $\mathbf{f}$ and $\mathbf{f'}$ be two flows in $\mathbf{G}$ with $\mathbf{v(f')} \geq \mathbf{v(f)}$. Then there is a flow $\mathbf{f''}$ of value $\mathbf{v(f')} - \mathbf{v(f)}$ in $\mathbf{G_f}$.*

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

# Residual Graph Property: Implication

*Recursive* algorithm for finding a maximum flow:

MaxFlow($\mathbf{G}, \mathbf{s}, \mathbf{t}$):
    If the flow from $\mathbf{s}$ to $\mathbf{t}$ is $\mathbf{0}$
        return $\mathbf{0}$
    Find any flow $\mathbf{f}$ with $\mathbf{v(f) > 0}$ in $\mathbf{G}$
    Recursively compute a maximum flow $\mathbf{f'}$ in $\mathbf{G_f}$
    Output the flow $\mathbf{f + f'}$

*Iterative* algorithm for finding a maximum flow:

MaxFlow($\mathbf{G}, \mathbf{s}, \mathbf{t}$):
    Start with flow $\mathbf{f}$ that is $\mathbf{0}$ on all edges
    While there is a flow $\mathbf{f'}$ in $\mathbf{G_f}$ with $\mathbf{v(f') > 0}$ do
        $\mathbf{f = f + f'}$
        Update $\mathbf{G_f}$
    endWhile
    Output $\mathbf{f}$

# Residual Graph Property: Implication

*Recursive* algorithm for finding a maximum flow:

MaxFlow($\mathbf{G}, \mathbf{s}, \mathbf{t}$):
    If the flow from $\mathbf{s}$ to $\mathbf{t}$ is $\mathbf{0}$
        return $\mathbf{0}$
    Find any flow $\mathbf{f}$ with $\mathbf{v(f)} > \mathbf{0}$ in $\mathbf{G}$
    Recursively compute a maximum flow $\mathbf{f}'$ in $\mathbf{G_f}$
    Output the flow $\mathbf{f} + \mathbf{f}'$

*Iterative* algorithm for finding a maximum flow:

MaxFlow($\mathbf{G}, \mathbf{s}, \mathbf{t}$):
    Start with flow $\mathbf{f}$ that is $\mathbf{0}$ on all edges
    While there is a flow $\mathbf{f}'$ in $\mathbf{G_f}$ with $\mathbf{v(f')} > \mathbf{0}$ do
        $\mathbf{f}$ = $\mathbf{f} + \mathbf{f}'$
        Update $\mathbf{G_f}$
    endWhile
    Output $\mathbf{f}$

# Ford-Fulkerson Algorithm

**algFordFulkerson**

for every edge $\mathbf{e}$, $\mathbf{f(e) = 0}$
$\mathbf{G_f}$ is residual graph of $\mathbf{G}$ with respect to $\mathbf{f}$
**while** $\mathbf{G_f}$ has a simple $\mathbf{s}$-$\mathbf{t}$ path **do**
  let $\mathbf{P}$ be simple $\mathbf{s}$-$\mathbf{t}$ path in $\mathbf{G_f}$
  $\mathbf{f} = \mathbf{augment(f, P)}$
  Construct new residual graph $\mathbf{G_f}$

**augment**$(\mathbf{f}, \mathbf{P})$

let $\mathbf{b}$ be bottleneck capacity,
  i.e., min capacity of edges in $\mathbf{P}$ (in $\mathbf{G_f}$)
**for** each edge $(\mathbf{u}, \mathbf{v})$ in $\mathbf{P}$ **do**
  **if** $\mathbf{e} = (\mathbf{u}, \mathbf{v})$ is a forward edge **then**
    $\mathbf{f(e) = f(e) + b}$
  **else** (* $(\mathbf{u}, \mathbf{v})$ is a backward edge *)
    let $\mathbf{e} = (\mathbf{v}, \mathbf{u})$ (* $(\mathbf{v}, \mathbf{u})$ is in $\mathbf{G}$ *)
    $\mathbf{f(e) = f(e) - b}$
**return** $\mathbf{f}$

# Ford-Fulkerson Algorithm

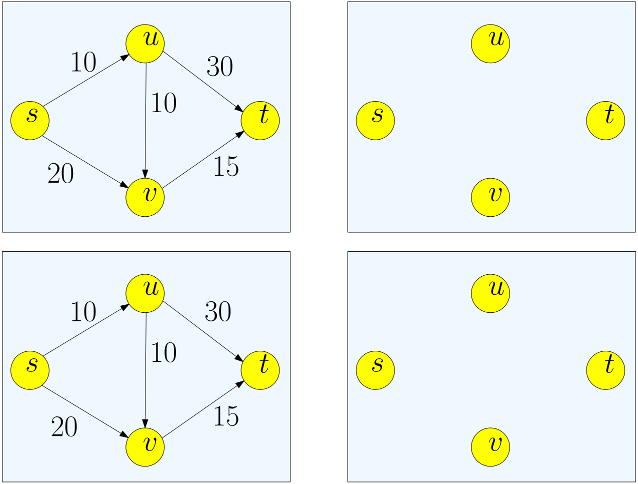
**algFordFulkerson**

```
for every edge e, f(e) = 0
G_f is residual graph of G with respect to f
while G_f has a simple s-t path do
    let P be simple s-t path in G_f
    f = augment(f, P)
    Construct new residual graph G_f
```

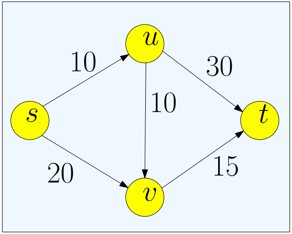
**augment**(f,P)

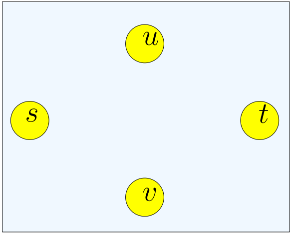
```
let b be bottleneck capacity,
    i.e., min capacity of edges in P (in G_f)
for each edge (u, v) in P do
    if e = (u, v) is a forward edge then
        f(e) = f(e) + b
    else (* (u, v) is a backward edge *)
        let e = (v, u) (* (v, u) is in G *)
        f(e) = f(e) − b
return f
```

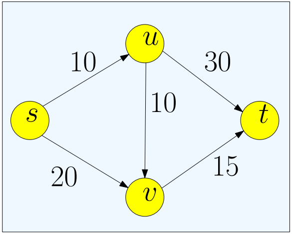
# Example

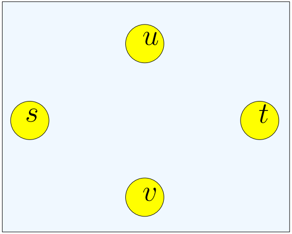


# Example continued

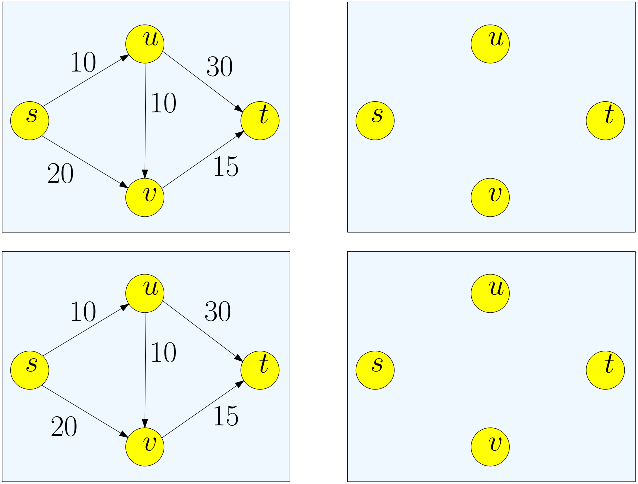


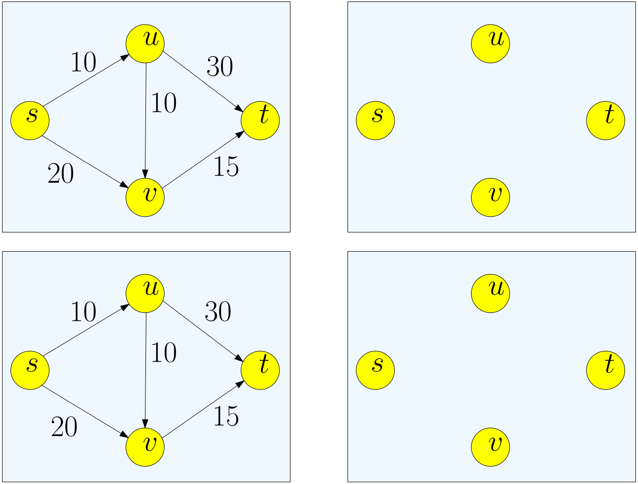



# Example continued



# Example continued



# Properties about Augmentation: Flow

## Lemma

*If $\mathbf{f}$ is a flow and $\mathbf{P}$ is a simple $\mathbf{s}$-$\mathbf{t}$ path in $\mathbf{G_f}$, then $\mathbf{f' = \mathtt{augment}(f, P)}$ is also a flow.*

## Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

- Capacity constraint: If $\mathbf{(u, v) \in P}$ is a forward edge then $\mathbf{f'(e) = f(e) + b}$ and $\mathbf{b \leq c(e) - f(e)}$. If $\mathbf{(u, v) \in P}$ is a backward edge, then letting $\mathbf{e = (v, u)}$, $\mathbf{f'(e) = f(e) - b}$ and $\mathbf{b \leq f(e)}$. Both cases $\mathbf{0 \leq f'(e) \leq c(e)}$.
- Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1, e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1, e_2}$ are forward or backward edges. Check cases (see fig next slide). $\square$

# Properties about Augmentation: Flow

## Lemma

*If* $\mathbf{f}$ *is a flow and* $\mathbf{P}$ *is a simple* $\mathbf{s}$*-*$\mathbf{t}$ *path in* $\mathbf{G_f}$*, then* $\mathbf{f'} = \mathtt{augment}(\mathbf{f}, \mathbf{P})$ *is also a flow.*

## Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

- Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f'(e)} = \mathbf{f(e)} + \mathbf{b}$ and $\mathbf{b} \leq \mathbf{c(e)} - \mathbf{f(e)}$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e} = (\mathbf{v}, \mathbf{u})$, $\mathbf{f'(e)} = \mathbf{f(e)} - \mathbf{b}$ and $\mathbf{b} \leq \mathbf{f(e)}$. Both cases $\mathbf{0} \leq \mathbf{f'(e)} \leq \mathbf{c(e)}$.
- Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1}, \mathbf{e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1}, \mathbf{e_2}$ are forward or backward edges. Check cases (see fig next slide). □

# Properties about Augmentation: Flow

## Lemma

*If $\mathbf{f}$ is a flow and $\mathbf{P}$ is a simple $\mathbf{s}$-$\mathbf{t}$ path in $\mathbf{G_f}$, then $\mathbf{f'} = \texttt{augment}(\mathbf{f}, \mathbf{P})$ is also a flow.*

## Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

- Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f'(e)} = \mathbf{f(e)} + \mathbf{b}$ and $\mathbf{b} \leq \mathbf{c(e)} - \mathbf{f(e)}$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e} = (\mathbf{v}, \mathbf{u})$, $\mathbf{f'(e)} = \mathbf{f(e)} - \mathbf{b}$ and $\mathbf{b} \leq \mathbf{f(e)}$. Both cases $\mathbf{0} \leq \mathbf{f'(e)} \leq \mathbf{c(e)}$.
- Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1}, \mathbf{e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1}, \mathbf{e_2}$ are forward or backward edges. Check cases (see fig next slide). □

# Properties about Augmentation: Flow

## Lemma

*If* $\mathbf{f}$ *is a flow and* $\mathbf{P}$ *is a simple* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$*, then* $\mathbf{f}' = \mathtt{augment}(\mathbf{f}, \mathbf{P})$ *is also a flow.*

## Proof.

Verify that $\mathbf{f}'$ is a flow. Let $\mathbf{b}$ be augmentation amount.

- Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f}'(\mathbf{e}) = \mathbf{f}(\mathbf{e}) + \mathbf{b}$ and $\mathbf{b} \leq \mathbf{c}(\mathbf{e}) - \mathbf{f}(\mathbf{e})$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e} = (\mathbf{v}, \mathbf{u})$, $\mathbf{f}'(\mathbf{e}) = \mathbf{f}(\mathbf{e}) - \mathbf{b}$ and $\mathbf{b} \leq \mathbf{f}(\mathbf{e})$. Both cases $\mathbf{0} \leq \mathbf{f}'(\mathbf{e}) \leq \mathbf{c}(\mathbf{e})$.
- Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e}_1, \mathbf{e}_2$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e}_1, \mathbf{e}_2$ are forward or backward edges. Check cases (see fig next slide). □

# Properties about Augmentation: Flow

## Lemma

*If* $\mathbf{f}$ *is a flow and* $\mathbf{P}$ *is a simple* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$, *then* $\mathbf{f'} = \mathtt{augment}(\mathbf{f}, \mathbf{P})$ *is also a flow.*

## Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

- Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f'(e) = f(e) + b}$ and $\mathbf{b \leq c(e) - f(e)}$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e = (v, u)}$, $\mathbf{f'(e) = f(e) - b}$ and $\mathbf{b \leq f(e)}$. Both cases $\mathbf{0 \leq f'(e) \leq c(e)}$.
- Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1, e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1, e_2}$ are forward or backward edges. Check cases (see fig next slide). □

# Properties about Augmentation: Conservation Constraint

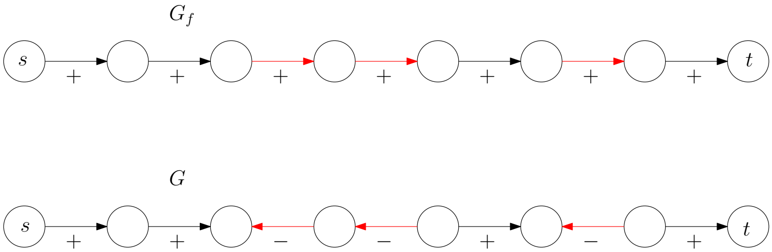



Figure: Augmenting path **P** in $\mathbf{G_f}$ and corresponding change of flow in **G**. Red edges are backward edges.

# Properties about Augmentation: Integer Flow

## Lemma

*At every stage of the Ford-Fulkerson algorithm, the flow values* $\mathbf{f(e)}$ *and the residual capacities in* $\mathbf{G_f}$ *are integers*

## Proof.

Initial flow and residual capacities are integers. Suppose lemma holds for $\mathbf{j}$ iterations. Then in $\mathbf{(j+1)}$st iteration, minimum capacity edge $\mathbf{b}$ is an integer, and so flow after augmentation is an integer. □

# Progress in Ford-Fulkerson

## Proposition

*Let* $\mathbf{f}$ *be a flow and* $\mathbf{f'}$ *be flow after one augmentation. Then* $\mathbf{v(f)} < \mathbf{v(f')}$.

## Proof.

Let $\mathbf{P}$ be an augmenting path, i.e., $\mathbf{P}$ is a simple $\mathbf{s}$-$\mathbf{t}$ path in residual graph

- First edge $\mathbf{e}$ in $\mathbf{P}$ must leave $\mathbf{s}$
- Original network $\mathbf{G}$ has no incoming edges to $\mathbf{s}$; hence $\mathbf{e}$ is a forward edge
- $\mathbf{P}$ is simple and so never returns to $\mathbf{s}$
- Thus, value of flow increases by the flow on edge $\mathbf{e}$ $\square$

# Termination Proof

## Theorem

*Let* $\mathbf{C}$ *be the minimum cut value; in particular* $\mathbf{C} \leq \sum_{\mathbf{e} \text{ out of } \mathbf{s}} \mathbf{c(e)}$*. Ford-Fulkerson algorithm terminates after finding at most* $\mathbf{C}$ *augmenting paths.*

## Proof.

The value of the flow increases by at least $\mathbf{1}$ after each augmentation. Maximum value of flow is at most $\mathbf{C}$. □

## Running time

- Number of iterations $\leq \mathbf{C}$
- Number of edges in $\mathbf{G_f} \leq \mathbf{2m}$
- Time to find augmenting path is $\mathbf{O(n+m)}$
- Running time is $\mathbf{O(C(n+m))}$ (or $\mathbf{O(mC)}$).

# Efficiency of Ford-Fulkerson

Running time $= \mathbf{O(mC)}$ is not polynomial. Can the running time be as $\mathbf{\Omega(mC)}$ or is our analysis weak?



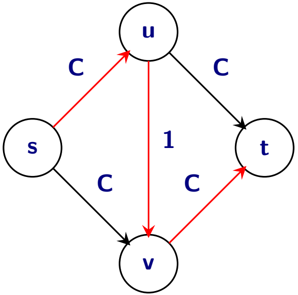




Ford-Fulkerson can take $\mathbf{\Omega(C)}$ iterations.

# Efficiency of Ford-Fulkerson

Running time $= \mathbf{O(mC)}$ is not polynomial. Can the running time be as $\mathbf{\Omega(mC)}$ or is our analysis weak?



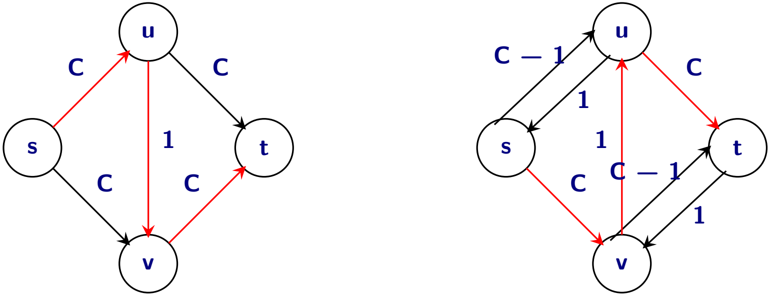




Ford-Fulkerson can take $\mathbf{\Omega(C)}$ iterations.

# Efficiency of Ford-Fulkerson

Running time $= O(mC)$ is not polynomial. Can the running time be as $\Omega(mC)$ or is our analysis weak?




Ford-Fulkerson can take $\Omega(C)$ iterations.

# Correctness of Ford-Fulkerson Augmenting Path Algorithm

Question: When the algorithm terminates, is the flow computed the maximum **s**-**t** flow?

Proof idea: show a cut of value equal to the flow. Also shows that maximum flow is equal to minimum cut!

# Correctness of Ford-Fulkerson Augmenting Path Algorithm

Question: When the algorithm terminates, is the flow computed the maximum **s**-**t** flow?

Proof idea: show a cut of value equal to the flow. Also shows that maximum flow is equal to minimum cut!

# Recalling Cuts

## Definition

Given a flow network an $\mathbf{s}$-$\mathbf{t}$ cut is a set of edges $\mathbf{E}' \subset \mathbf{E}$ such that removing $\mathbf{E}'$ *disconnects* $\mathbf{s}$ from $\mathbf{t}$: in other words there is no directed $\mathbf{s} \rightarrow \mathbf{t}$ path in $\mathbf{E} - \mathbf{E}'$. *Capacity* of cut $\mathbf{E}'$ is $\sum_{\mathbf{e} \in \mathbf{E}'} \mathbf{c}(\mathbf{e})$.

Let $\mathbf{A} \subset \mathbf{V}$ such that

- $\mathbf{s} \in \mathbf{A}$, $\mathbf{t} \notin \mathbf{A}$
- $\mathbf{B} = \mathbf{V} - \mathbf{A}$ and hence $\mathbf{t} \in \mathbf{B}$

Define $(\mathbf{A}, \mathbf{B}) = \{(\mathbf{u}, \mathbf{v}) \in \mathbf{E} \mid \mathbf{u} \in \mathbf{A}, \mathbf{v} \in \mathbf{B}\}$

## Claim

$(\mathbf{A}, \mathbf{B})$ *is an* $\mathbf{s}$-$\mathbf{t}$ *cut.*

Recall: Every *minimal* $\mathbf{s}$-$\mathbf{t}$ cut $\mathbf{E}'$ is a cut of the form $(\mathbf{A}, \mathbf{B})$.

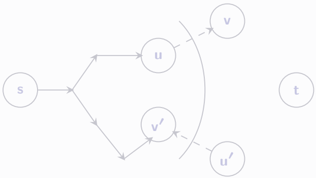
# Ford-Fulkerson Correctness

## Lemma

*If there is no* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$ *then there is some cut* $\mathbf{(A, B)}$ *such that* $\mathbf{v(f) = c(A, B)}$

## Proof.

Let $\mathbf{A}$ be all vertices reachable from $\mathbf{s}$ in $\mathbf{G_f}$; $\mathbf{B = V \setminus A}$




- $\mathbf{s \in A}$ and $\mathbf{t \in B}$. So $\mathbf{(A, B)}$ is an $\mathbf{s}$-$\mathbf{t}$ cut in $\mathbf{G}$
- If $\mathbf{e = (u, v) \in G}$ with $\mathbf{u \in A}$ and $\mathbf{v \in B}$, then $\mathbf{f(e) = c(e)}$ (saturated edge) because otherwise $\mathbf{v}$ is reachable from $\mathbf{s}$ in $\mathbf{G_f}$

□

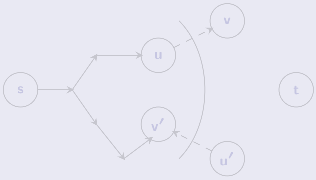
# Ford-Fulkerson Correctness

## Lemma

*If there is no* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$ *then there is some cut* $\mathbf{(A, B)}$ *such that* $\mathbf{v(f) = c(A, B)}$

## Proof.

Let $\mathbf{A}$ be all vertices reachable from $\mathbf{s}$ in $\mathbf{G_f}$; $\mathbf{B = V \setminus A}$




- $\mathbf{s \in A}$ and $\mathbf{t \in B}$. So $\mathbf{(A, B)}$ is an $\mathbf{s}$-$\mathbf{t}$ cut in $\mathbf{G}$
- If $\mathbf{e = (u, v) \in G}$ with $\mathbf{u \in A}$ and $\mathbf{v \in B}$, then $\mathbf{f(e) = c(e)}$ (saturated edge) because otherwise $\mathbf{v}$ is reachable from $\mathbf{s}$ in $\mathbf{G_f}$

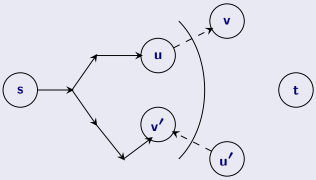
# Ford-Fulkerson Correctness

## Lemma

*If there is no* $\mathbf{s}$-$\mathbf{t}$ *path in* $G_f$ *then there is some cut* $(A, B)$ *such that* $v(f) = c(A, B)$

## Proof.

Let $A$ be all vertices reachable from $s$ in $G_f$; $B = V \setminus A$




- $s \in A$ and $t \in B$. So $(A, B)$ is an $s$-$t$ cut in $G$
- If $e = (u, v) \in G$ with $u \in A$ and $v \in B$, then $f(e) = c(e)$ (saturated edge) because otherwise $v$ is reachable from $s$ in $G_f$

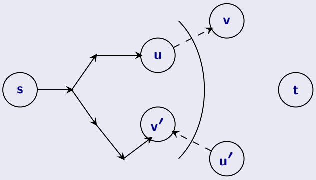
# Ford-Fulkerson Correctness

## Lemma

*If there is no* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$ *then there is some cut* $\mathbf{(A, B)}$ *such that* $\mathbf{v(f) = c(A, B)}$

## Proof.

Let $\mathbf{A}$ be all vertices reachable from $\mathbf{s}$ in $\mathbf{G_f}$; $\mathbf{B = V \setminus A}$

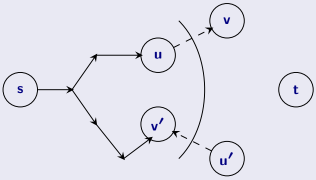



- $\mathbf{s \in A}$ and $\mathbf{t \in B}$. So $\mathbf{(A, B)}$ is an $\mathbf{s}$-$\mathbf{t}$ cut in $\mathbf{G}$
- If $\mathbf{e = (u, v) \in G}$ with $\mathbf{u \in A}$ and $\mathbf{v \in B}$, then $\mathbf{f(e) = c(e)}$ (saturated edge) because otherwise $\mathbf{v}$ is reachable from $\mathbf{s}$ in $\mathbf{G_f}$

□

# Lemma Proof Continued

## Proof.

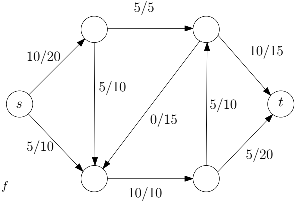



- If $e = (u', v') \in G$ with $u' \in B$ and $v' \in A$, then $f(e) = 0$ because otherwise $u'$ is reachable from $s$ in $G_f$
- Thus,

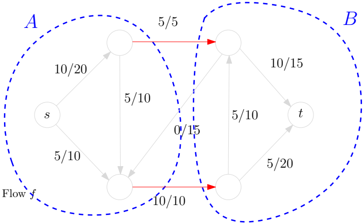
$$
\begin{aligned}
v(f) &= f^{out}(A) - f^{in}(A) \\
&= f^{out}(A) - 0 \\
&= c(A, B) - 0 \\
&= c(A, B)
\end{aligned}
$$

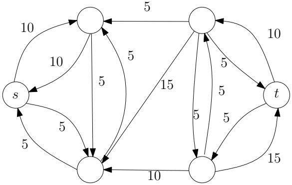
$\square$

# Example

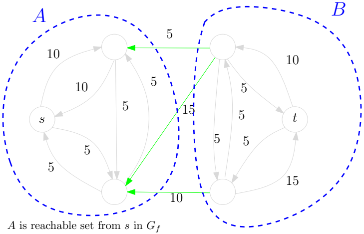



Flow $f$



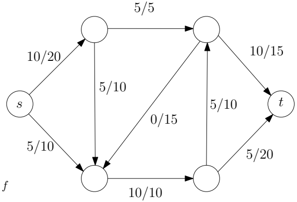




Residual graph $G_f$: no $s$-$t$ path

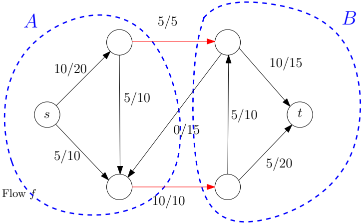



$A$ is reachable set from $s$ in $G_f$

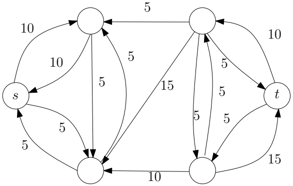
# Example



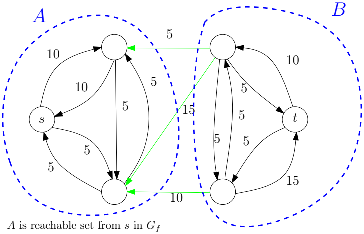

Flow $f$




Flow $f$




Residual graph $G_f$: no $s$-$t$ path




$A$ is reachable set from $s$ in $G_f$

# Ford-Fulkerson Correctness

## Theorem

*The flow returned by the algorithm is the maximum flow.*

## Proof.

- For any flow $\mathbf{f}$ and $\mathbf{s}$-$\mathbf{t}$ cut $(\mathbf{A}, \mathbf{B})$, $\mathbf{v}(\mathbf{f}) \leq \mathbf{c}(\mathbf{A}, \mathbf{B})$
- For flow $\mathbf{f}^*$ returned by algorithm, $\mathbf{v}(\mathbf{f}^*) = \mathbf{c}(\mathbf{A}^*, \mathbf{B}^*)$ for some $\mathbf{s}$-$\mathbf{T}$ cut $(\mathbf{A}^*, \mathbf{B}^*)$
- Hence, $\mathbf{f}^*$ is maximum

$\square$

# Max-Flow Min-Cut Theorem and Integrality of Flows

## Theorem

*For any network* $\mathbf{G}$*, the value of a maximum* $\mathbf{s}$-$\mathbf{t}$ *flow is equal to the capacity of the minimum* $\mathbf{s}$-$\mathbf{t}$ *cut.*

## Proof.

Ford-Fulkerson algorithm terminates with a maximum flow of value equal to the capacity of a (minimum) cut. $\square$

# Max-Flow Min-Cut Theorem and Integrality of Flows

## Theorem

*For any network* $\mathbf{G}$ *with integer capacities, there is a maximum* $\mathbf{s}$-$\mathbf{t}$ *flow that is integer valued.*

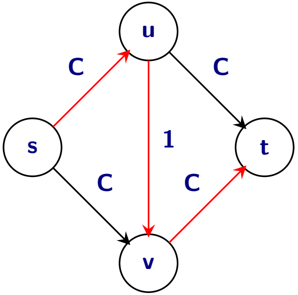
## Proof.

Ford-Fulkerson algorithm produces an integer valued flow when capacities are integers. □

# Efficiency of Ford-Fulkerson

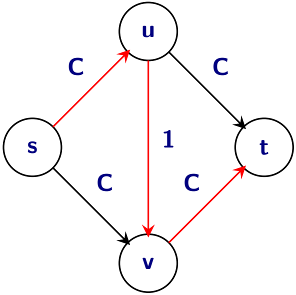
Running time $= \mathbf{O(mC)}$ is not polynomial. Can the upper bound be achieved?

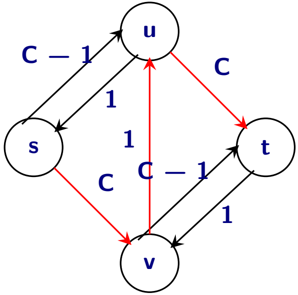
Running time $= \mathbf{O(mC)}$ is not polynomial. Can the upper bound be achieved?



# Efficiency of Ford-Fulkerson

Running time $= \mathbf{O(mC)}$ is not polynomial. Can the upper bound be achieved?



# Polynomial Time Algorithms

Question: Is there a polynomial time algorithm for maxflow?

Question: Is there a variant of Ford-Fulkerson that leads to a polynomial time algorithm? Can we choose an augmenting path in some clever way? Yes! Two variants.

- Choose the augmenting path with largest bottleneck capacity.
- Choose the shortest augmenting path.

# Polynomial Time Algorithms

Question: Is there a polynomial time algorithm for maxflow?

Question: Is there a variant of Ford-Fulkerson that leads to a polynomial time algorithm? Can we choose an augmenting path in some clever way? Yes! Two variants.

- Choose the augmenting path with largest bottleneck capacity.
- Choose the shortest augmenting path.

# Polynomial Time Algorithms

Question: Is there a polynomial time algorithm for maxflow?

Question: Is there a variant of Ford-Fulkerson that leads to a polynomial time algorithm? Can we choose an augmenting path in some clever way? Yes! Two variants.

- Choose the augmenting path with largest bottleneck capacity.
- Choose the shortest augmenting path.

# Augmenting Paths with Large Bottleneck Capacity

- Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson
- How do we find path with largest bottleneck capacity?
  - Assume we know $\Delta$ the bottleneck capacity
  - Remove all edges with residual capacity $\leq \Delta$
  - Check if there is a path from $s$ to $t$
  - Do binary search to find largest $\Delta$
  - Running time: $O(m \log C)$
- Can we bound the number of augmentations? Can show that in $O(m \log C)$ augmentations the algorithm reaches a max flow. This leads to an $O(m^2 \log^2 C)$ time algorithm.

# Augmenting Paths with Large Bottleneck Capacity

- Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson
- How do we find path with largest bottleneck capacity?
  - Assume we know $\Delta$ the bottleneck capacity
  - Remove all edges with residual capacity $\leq \Delta$
  - Check if there is a path from $s$ to $t$
  - Do binary search to find largest $\Delta$
  - Running time: $O(m \log C)$
- Can we bound the number of augmentations? Can show that in $O(m \log C)$ augmentations the algorithm reaches a max flow. This leads to an $O(m^2 \log^2 C)$ time algorithm.

# Augmenting Paths with Large Bottleneck Capacity

- Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson
- How do we find path with largest bottleneck capacity?
  - Assume we know $\Delta$ the bottleneck capacity
  - Remove all edges with residual capacity $\leq \Delta$
  - Check if there is a path from $s$ to $t$
  - Do binary search to find largest $\Delta$
  - Running time: $O(m \log C)$
- Can we bound the number of augmentations? Can show that in $O(m \log C)$ augmentations the algorithm reaches a max flow. This leads to an $O(m^2 \log^2 C)$ time algorithm.

# Augmenting Paths with Large Bottleneck Capacity

How do we find path with largest bottleneck capacity?

- Max bottleneck capacity is one of the edge capacities. Why?
- Can do binary search on the edge capacities. First, sort the edges by their capacities and then do binary search on that array as before.
- Algorithm's running time is $\mathbf{O(m \log m)}$.
- Different algorithm that also leads to $\mathbf{O(m \log m)}$ time algorithm by adapting Prim's algorithm.

# Removing Dependence on $C$

- [Edmonds-Karp, Dinitz] Picking augmenting paths with fewest number of edges yields a $O(m^2n)$ algorithm, i.e., independent of $C$. Such an algorithm is called a strongly polynomial time algorithm since the running time does not depend on the numbers (assuming RAM model). (Many implementation of Ford-Fulkerson would actually use shortest augmenting path if they use BFS to find an $s$-$t$ path).
- Further improvements can yield algorithms running in $O(mn \log n)$, or $O(n^3)$.

# Finding a Minimum Cut

Question: How do we find an actual minimum $\mathbf{s}$-$\mathbf{t}$ cut?

Proof gives the algorithm!

- Compute an $\mathbf{s}$-$\mathbf{t}$ maximum flow $\mathbf{f}$ in $\mathbf{G}$
- Obtain the residual graph $\mathbf{G_f}$
- Find the nodes $\mathbf{A}$ reachable from $\mathbf{s}$ in $\mathbf{G_f}$
- Output the cut $(\mathbf{A}, \mathbf{B}) = \{(\mathbf{u}, \mathbf{v}) \mid \mathbf{u} \in \mathbf{A}, \mathbf{v} \in \mathbf{B}\}$. Note: The cut is found in $\mathbf{G}$ while $\mathbf{A}$ is found in $\mathbf{G_f}$

Running time is essentially the same as finding a maximum flow.

Note: Given $\mathbf{G}$ and a flow $\mathbf{f}$ there is a linear time algorithm to check if $\mathbf{f}$ is a maximum flow and if it is, outputs a minimum cut. How?

# Finding a Minimum Cut

Question: How do we find an actual minimum $\mathbf{s}$-$\mathbf{t}$ cut?
Proof gives the algorithm!

- Compute an $\mathbf{s}$-$\mathbf{t}$ maximum flow $\mathbf{f}$ in $\mathbf{G}$
- Obtain the residual graph $\mathbf{G_f}$
- Find the nodes $\mathbf{A}$ reachable from $\mathbf{s}$ in $\mathbf{G_f}$
- Output the cut $(\mathbf{A}, \mathbf{B}) = \{(\mathbf{u}, \mathbf{v}) \mid \mathbf{u} \in \mathbf{A}, \mathbf{v} \in \mathbf{B}\}$. Note: The cut is found in $\mathbf{G}$ while $\mathbf{A}$ is found in $\mathbf{G_f}$

Running time is essentially the same as finding a maximum flow.

Note: Given $\mathbf{G}$ and a flow $\mathbf{f}$ there is a linear time algorithm to check if $\mathbf{f}$ is a maximum flow and if it is, outputs a minimum cut. How?

# Notes

# Notes

# Notes

# Notes