Flow Networks

CSCI 532
Motivation

Suppose we have a directed graph that represent an oil pipeline network. Edge weight represent pipe capacity. How much oil can we transfer from source $s$ to sink $t$?
Motivation

Suppose we have a directed graph that represent an oil pipeline network. Edge weight represent pipe capacity. How much oil can we transfer from source $s$ to sink $t$?
Motivation

Suppose we have a directed graph that represents an oil pipeline network. Edge weight represents pipe capacity. How much oil can we transfer from source $s$ to sink $t$?

Flow networks are commonly used to model transportation networks:
- Edges carry traffic (capacity constrained).
- Nodes act as junctions between edges.

Examples:
- Internet routing.
- Road networks.
- Electricity distribution.
Flow Network

Flow Network:

An $s - t$ flow is a function $f: E \rightarrow \mathbb{R}^+$ such that:
Flow Network

Flow Network:
• Directed-edge graph, $G = (V, E)$.
• Non-negative edge capacity, $c_e$.
• Single source, $s$, without input edges.
• Single sink, $t$, without output edges.

An $s \rightarrow t$ flow is a function $f: E \rightarrow \mathbb{R}^+$ such that:
Flow Network

Flow Network:
- Directed-edge graph, $G = (V, E)$.
- Non-negative edge capacity, $c_e$.
- Single source, $s$, without input edges.
- Single sink, $t$, without output edges.

An $s - t$ flow is a function $f: E \to \mathbb{R}^+$ such that:
- $0 \leq f(e) \leq c_e, \forall e \in E$. (capacity constraint)
Flow Network

Flow Network:

- Directed-edge graph, $G = (V, E)$.
- Non-negative edge capacity, $c_e$.
- Single source, $s$, without input edges.
- Single sink, $t$, without output edges.

An $s - t$ flow is a function $f : E \to \mathbb{R}^+$ such that:

- $0 \leq f(e) \leq c_e, \forall e \in E$. (capacity constraint)
- $\sum_{e \in \text{input}(v)} f(e) = \sum_{e \in \text{output}(v)} f(e), \forall v \in V \setminus \{s, t\}$. (conservation of flow constraint: “Everything that goes into a node has to come out, except for $s$ and $t$”)

Flow Network

Flow Network:
- Directed-edge graph, $G = (V, E)$.
- Non-negative edge capacity, $c_e$.
- Single source, $s$, without input edges.
- Single sink, $t$, without output edges.

An $s - t$ flow is a function $f: E \to \mathbb{R}^+$ such that:
- $0 \leq f(e) \leq c_e$, $\forall e \in E$. (capacity constraint)
- $\sum_{e \in \text{input}(v)} f(e) = \sum_{e \in \text{output}(v)} f(e)$, $\forall v \in V \setminus \{s, t\}$.
  (conservation of flow constraint: “Everything that goes into a node has to come out, except for $s$ and $t$”)
- Value of flow $= \text{val}(f) = \sum_{e \in \text{output}(s)} f(e) = \sum_{e \in \text{input}(t)} f(e)$
Flow Network

Flow Network:
• Directed-edge graph, $G = (V, E)$.
• Non-negative edge capacity, $c_e$.
• Single source, $s$, without input edges.
• Single sink, $t$, without output edges.

An $s - t$ flow is a function $f : E \rightarrow \mathbb{R}^+$ such that:
• $0 \leq f(e) \leq c_e, \forall e \in E$. (capacity constraint)
• $\sum_{e \in \text{input}(v)} f(e) = \sum_{e \in \text{output}(v)} f(e), \forall v \in V \setminus \{s, t\}$.  
  (conservation of flow constraint: “Everything that goes into a node has to come out, except for $s$ and $t$”)
• Value of flow = $val(f) = \sum_{e \in \text{output}(s)} f(e) = \sum_{e \in \text{input}(t)} f(e)$

Maximum Flow Problem:
Given a flow network, find the maximum possible value of flow.
Maximum Flow Algorithm

Can we use Dynamic Programming?

Can we use a Greedy approach?
Maximum Flow Algorithm

Can we use Dynamic Programming?
What is the optimal substructure?
Can we use a Greedy approach?
What would the greedy choice be?
Maximum Flow Algorithm

Can we use Dynamic Programming?
   What is the optimal substructure?
Can we use a Greedy approach?
   What would the greedy choice be?

Greedy Choice: Select path that can handle most flow, update capacities, repeat.
Maximum Flow Algorithm

Can we use Dynamic Programming?
What is the optimal substructure?
Can we use a Greedy approach?
What would the greedy choice be?

Greedy Choice: Select path that can handle most flow, update capacities, repeat.

Flow of 20, but max flow is 30. Need way to “undo” parts of previous decisions.
Residual Graph

Residual Graph: Tool to re-route flow we already decided to push.
Residual Graph

Given a flow network $G$, and a flow $f$, define the residual graph $G_f$ as:

- Identical nodes as $G$. 

![Graph Diagram](image-url)
Residual Graph

Given a flow network $G$, and a flow $f$, define the residual graph $G_f$ as:

- Identical nodes as $G$.
- For each edge $e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
Residual Graph

Given a flow network $G$, and a flow $f$, define the residual graph $G_f$ as:

- Identical nodes as $G$.
- For each edge $e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- For each edge $e = (u, v)$, if $f(e) > 0$, create new edge $e' = (v, u)$ with $c_{e'} = f(e)$ ($e'$ called back edge).
Flows in Residual Graphs

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

$P = ?$
Flows in Residual Graphs

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

\[
P = s \rightarrow b \rightarrow a \rightarrow t
\]
Flows in Residual Graphs

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

$bottleneck(P, f) = \text{minimum residual capacity on any edge in } P = 10$. 

$P = s \rightarrow b \rightarrow a \rightarrow t$
Flows in Residual Graphs

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

\[
\text{bottleneck}(P, f) = \text{minimum residual capacity on any edge in } P = 10.
\]

\[
\text{augment}(f, P)
\]

\[
b = \text{bottleneck}(P, f) = 10
\]

\[
\text{for each edge } (u, v) \text{ in } P
\]

\[
\text{if } (u, v) \text{ is a back edge } f((v, u)) = b
\]

\[
\text{else } f((u, v)) += b
\]

\[
\text{return } f
\]
Flows in Residual Graphs

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

\[
\text{bottleneck}(P, f) = \text{minimum residual capacity on any edge in } P = 10.
\]

\[
\text{augment}(f, P)
\]

\[
b = \text{bottleneck}(P, f) = 10
\]

\[
\text{for each edge } (u, v) \text{ in } P
\]

\[
\text{if } (u, v) \text{ is a back edge}
\]

\[
f((v, u)) -= b
\]

\[
\text{else}
\]

\[
f((u, v)) += b
\]

\[
\text{return } f
\]

\[
P = s \rightarrow b \rightarrow a \rightarrow t
\]
Flows in Residual Graphs

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

$bottleneck(P, f) = \text{minimum residual capacity on any edge in } P = 10.$

$augment(f, P)$

$b = bottleneck(P, f) = 10$  
for each edge $(u, v)$ in $P$  
if $(u, v)$ is a back edge  
$f((v, u)) = b$  
else  
$f((u, v)) += b$  
return $f$
Flows in Residual Graphs

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

$bottleneck(P, f) = minimum residual capacity on any edge in P = 10.$

augment($f$, $P$)
  $b = bottleneck(P, f) = 10$
  for each edge $(u, v)$ in $P$
    if $(u, v)$ is a back edge
      $f((v, u)) = b$
    else
      $f((u, v)) += b$
  return $f$

$P = s \rightarrow b \rightarrow a \rightarrow t$
Flows in Residual Graphs

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

$bottleneck(P, f) = \text{minimum residual capacity on any edge in } P = 10.$

$augment(f, P)
\quad b = bottleneck(P, f) = 10
\quad \text{for each edge } (u, v) \text{ in } P
\quad \quad \text{if } (u, v) \text{ is a back edge}
\quad \quad \quad f((v, u)) = b
\quad \text{else}
\quad \quad f((u, v)) = b
\quad \text{return } f$

$P = s \rightarrow b \rightarrow a \rightarrow t$
Flows in Residual Graphs

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

\[ \text{bottleneck}(P, f) = \text{minimum residual capacity on any edge in } P = 10. \]

\[ \text{augment}(f, P) \]
\[ b = \text{bottleneck}(P, f) = 10 \]
\[ \text{for each edge } (u, v) \text{ in } P \]
\[ \text{if } (u, v) \text{ is a back edge } \]
\[ f((v, u)) = b \]
\[ \text{else } \]
\[ f((u, v)) += b \]
\[ \text{return } f \]
Let $P$ be a simple $s - t$ path in $G_f$.

$bottleneck(P, f) = \text{minimum residual capacity on any edge in } P = 10$.

$augment(f, P)$

\[
b = bottleneck(P, f) = 10
\]

\[
\text{for each edge } (u, v) \text{ in } P
\]

\[
\text{if } (u, v) \text{ is a back edge}
\]

\[
f((v, u)) = b
\]

\[
\text{else}
\]

\[
f((u, v)) += b
\]

\[
\text{return } f
\]
Ford-Fulkerson Algorithm

Max-Flow(G)
  \( f(e) = 0 \) for all \( e \) in \( G \)
  while simple s-t path in residual graph \( G_f \) exists
    \( P = \) simple s-t path in \( G_f \)
    \( f' = \) augment\( (f, P) \)
    \( f = f' \)
    \( G_f = G_f \)'
  return \( f \)

I.e. Push flow along \( P \).
Recalculate residual graph based on new flow \( f' \).
Ford-Fulkerson Algorithm

Max-Flow(G)
  \( f(e) = 0 \) for all \( e \) in \( G \)
  while \( s-t \) path in \( G_f \) exists
    \( P = \) simple \( s-t \) path in \( G_f \)
    \( f' = \) augment\( (f, P) \)
    \( f = f' \)
  \( G_f = G_f' \)
  return \( f \)

augment\( (f, P) \)
  \( b = \) bottleneck\( (P,f) \)
  for each edge \( (u,v) \) in \( P \)
    if \( (u,v) \) is a back edge
      \( f((v,u)) -= b \)
    else
      \( f((u,v)) += b \)
  return \( f \)

Residual graph for flow \( f, G_f \):
  \( * \) \( \forall e, \) if \( f(e) < c_e \), let \( c'_e = c_e - f(e) \).
  \( * \) \( \forall e = (u,v), \) if \( f(e) > 0 \), create \( e' = (v,u) \) with \( c'_e = f(e) \).
Ford-Fulkerson Algorithm

Max-Flow(G)
\[
f(e) = 0 \text{ for all } e \in G
\]
while s-t path in \( G_f \) exists
\[
P = \text{simple s-t path in } G_f
\]
f' = augment(f, P)
f = f'
\[
G_f = G_f'
\]
return f

augment(f, P)
\[
b = \text{bottleneck}(P, f)
\]
for each edge \((u, v)\) in P
\[
\text{if } (u, v) \text{ is a back edge}
\]
f\((v, u)\) -= b
\[
\text{else}
\]
f\((u, v)\) += b
return f

Residual graph for flow \( f, G_f \):
- \( \forall e \), if \( f(e) < c_e \), let \( c_e = c_e - f(e) \).
- \( \forall e = (u, v) \), if \( f(e) > 0 \), create \( e' = (v, u) \) with \( c_{e'} = f(e) \).
Ford-Fulkerson Algorithm

Max-Flow(G)

\[ f(e) = 0 \text{ for all } e \text{ in } G \]

while s-t path in \( G_f \) exists

\[ P = \text{simple s-t path in } G_f \]

\[ f' = \text{augment}(f, P) \]

\[ f = f' \]

\( G_f = G_{f'} \)

return \( f \)

augment(f, P)

\[ b = \text{bottleneck}(P, f) \]

for each edge \((u, v)\) in \( P\)

if \((u, v)\) is a back edge

\[ f((v, u)) -= b \]

else

\[ f((u, v)) += b \]

return \( f \)

Residual graph for flow \( f, G_f \):

\( \forall e, \text{if } f(e) < c_e, \text{let } c_e = c_e - f(e). \)

\( \forall e = (u, v), \text{if } f(e) > 0, \text{create } e' = (v, u) \text{ with } c_{e'} = f(e) \)
Ford-Fulkerson Algorithm

Max-Flow(G)
   \( f(e) = 0 \) for all \( e \) in \( G \)
while s-t path in \( G_f \) exists
   \( P = \) simple s-t path in \( G_f \)
   \( f' = \) augment\( (f, P) \)
   \( f = f' \)
\( G_f = G_f' \),
return \( f \)

augment\( (f, P) \)
   \( b = \) bottleneck\( (P, f) \)
for each edge \( (u, v) \) in \( P \)
   if \( (u, v) \) is a back edge
      \( f((v, u)) = b \)
   else
      \( f((u, v)) = b \)
return \( f \)

Residual graph for flow \( f, G_f \):
- \( \forall e, \) if \( f(e) < c_e \), let \( c_e = c_e - f(e) \).
- \( \forall e = (u, v), \) if \( f(e) > 0 \), create \( e' = (v, u) \) with \( c_{e'} = f(e) \)
Ford-Fulkerson Algorithm

Max-Flow(G)

\[ f(e) = 0 \text{ for all } e \text{ in } G \]

while s-t path in \( G_f \) exists

P = simple s-t path in \( G_f \)

\[ f' = \text{augment}(f, P) \]

\[ f = f' \]

\[ G_f = G_f' \]

return \( f \)

augment \( f, P \)

\[ b = \text{bottleneck}(P, f) \]

for each edge \((u, v)\) in \( P \)

if \((u, v)\) is a back edge

\[ f((v, u)) = -b \]

else

\[ f((u, v)) = +b \]

return \( f \)

Residual graph for flow \( f, G_f \):

- \( \forall e, if f(e) < c_e, let c_e = c_e - f(e) \).
- \( \forall e = (u, v), if f(e) > 0, create e' = (v, u) with c_{e'} = f(e) \)
Ford-Fulkerson Algorithm

Max-Flow(G)
\[ f(e) = 0 \text{ for all } e \text{ in } G \]
while s-t path in \( G_f \) exists
\[ P = \text{simple s-t path in } G_f \]
\[ f' = \text{augment}(f, P) \]
\[ f = f' \]
\[ G_f = G_{f'} \]
return \( f \)

augment(f, P)
\[ b = \text{bottleneck}(P,f) \]
for each edge \((u, v)\) in \( P \)
if \((u, v)\) is a back edge
\[ f((v, u)) -= b \]
else
\[ f((u, v)) += b \]
return \( f \)

Residual graph for flow \( f, G_f \):
- \( \forall e, \text{if } f(e) < c_e, \text{let } c_e = c_e - f(e) \).
- \( \forall e = (u, v), \text{if } f(e) > 0, \text{create } e' = (v, u) \text{ with } c_{e'} = f(e) \)
Ford-Fulkerson Algorithm

Max-Flow(G)
\[ f(e) = 0 \text{ for all } e \text{ in } G \]

while s-t path in G exists
\[ P = \text{simple s-t path in } G_f \]
\[ f' = \text{augment}(f, P) \]
\[ f = f' \]
\[ G_f = G_{f'} \]
return \( f \)

augment(f, P)
\[ b = \text{bottleneck}(P, f) \]
for each edge \((u, v)\) in \( P \)
\[ \text{if } (u, v) \text{ is a back edge} \]
\[ f((v, u)) =- b \]
\[ \text{else} \]
\[ f((u, v)) =+ b \]
return \( f \)
Ford-Fulkerson Algorithm

Max-Flow(G)
  \( f(e) = 0 \) for all \( e \) in \( G \)
  while s-t path in \( G_f \) exists
    P = simple s-t path in \( G_f \)
    \( f' = \text{augment}(f, P) \)
    \( f = f' \)
    \( G_f = G_f' \)
  return \( f \)

augment(f, P)
  \( b = \text{bottleneck}(P, f) \)
  for each edge \( (u, v) \) in \( P \)
    if \( (u, v) \) is a back edge
      \( f((v, u)) -= b \)
    else
      \( f((u, v)) += b \)
  return \( f \)

Residual graph for flow \( f, G_f \):
- \( \forall e, \text{if } f(e) < c_e, \text{let } c_e = c_e - f(e) \).
- \( \forall e = (u, v), \text{if } f(e) > 0, \text{create } e' = (v, u) \text{ with } c_{e'} = f(e) \)
Ford-Fulkerson Algorithm

Max-Flow(G)

\[ f(e) = 0 \text{ for all } e \text{ in } G \]

while s-t path in \( G_f \) exists

\[ P = \text{simple s-t path in } G_f \]

\[ f' = \text{augment}(f, P) \]

\[ f = f' \]

\[ G_f = G_{f'}, \]

return \( f \)

augment(f, P)

\[ b = \text{bottleneck}(P,f) \]

for each edge \((u, v)\) in \( P\)

if \((u, v)\) is a back edge

\[ f((v, u)) -= b \]

else

\[ f((u, v)) += b \]

return \( f \)

Residual graph for flow \( f \), \( G_f \):

- \( \forall e \), if \( f(e) < c_e \), let \( c_e = c_e - f(e) \).
- \( \forall e = (u, v) \), if \( f(e) > 0 \), create \( e' = (v, u) \) with \( c_{e'} = f(e) \)

Residual graph for flow \( f \), \( G_f \):

\[ g = \text{residual graph} \]

\[ P = s \rightarrow c \rightarrow b \rightarrow t \]

\[ G_f : \]

\[ G : \]

\[ f : \]
Ford-Fulkerson Algorithm

Max-Flow(G)

\[ f(e) = 0 \text{ for all } e \text{ in } G \]

while s-t path in \( G_f \) exists

1. \( P = \text{simple s-t path in } G_f \)
2. \( f' = \text{augment}(f, P) \)
3. \( f = f' \)

\( G_f = G_f' \)

return \( f \)

augment(f, P)

\[ b = \text{bottleneck}(P,f) \]

for each edge \((u, v)\) in \( P\)

1. if \((u, v)\) is a back edge
   \[ f((v, u)) -= b \]
2. else
   \[ f((u, v)) += b \]

return \( f \)

Residual graph for flow \( f, G_f \):
- \( \forall e, \text{if } f(e) < c_e, \text{let } c_e = c_e - f(e) \).
- \( \forall e = (u, v), \text{if } f(e) > 0, \text{create } e' = (v, u) \text{ with } c_e' = f(e) \).

\[ P = s \rightarrow a \rightarrow t \]
Ford-Fulkerson Algorithm

Max-Flow(G)
\[ f(e) = 0 \text{ for all } e \text{ in } G \]
while s-t path in \( G_f \) exists
\[ P = \text{simple s-t path in } G_f \]
\[ f' = \text{augment}(f, P) \]
\[ f = f' \]
\[ G_f = G_{f'} \]
return \( f \)

augment(f, P)
\[ b = \text{bottleneck}(P, f) \]
for each edge \((u, v)\) in \( P \)
\[ \text{if } (u, v) \text{ is a back edge} \]
\[ f((v, u)) -= b \]
else
\[ f((u, v)) += b \]
return \( f \)

Residual graph for flow \( f, G_f \):
- \( \forall e, \text{if } f(e) < c_e, \text{let } c_e = c_e - f(e) \).
- \( \forall e = (u, v), \text{if } f(e) > 0, \text{create } e' = (v, u) \text{ with } c_{e'} = f(e) \)
Ford-Fulkerson Algorithm

Max-Flow(G)
- $f(e) = 0$ for all $e$ in $G$
- while s-t path in $G_f$ exists
  - $P =$ simple s-t path in $G_f$
  - $f' = \text{augment}(f, P)$
  - $f = f'$
- $G_f = G_{f'}$
- return $f$

augment($f$, $P$)
- $b = \text{bottleneck}(P, f)$
- for each edge $(u, v)$ in $P$
  - if $(u, v)$ is a back edge
    - $f((v, u)) -= b$
  - else
    - $f((u, v)) += b$
- return $f$

Residual graph for flow $f$, $G_f$:
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$
Ford-Fulkerson Algorithm

Max-Flow(G)
   \( f(e) = 0 \) for all \( e \) in \( G \)
while s-t path in \( G_f \) exists
   P = simple s-t path in \( G_f \)
   \( f' = \text{augment}(f, P) \)
   \( f = f' \)
\( G_f = G_f' \)
return \( f \)

augment\( (f, P) \)
   \( b = \text{bottleneck}(P,f) \)
   for each edge \( (u, v) \) in \( P \)
      if \( (u, v) \) is a back edge
         \( f((v, u)) = -b \)
      else
         \( f((u, v)) = +b \)
   return \( f \)

Residual graph for flow \( f, G_f \):
   - \( \forall e, \text{if} \ f(e) < c_e, \text{let} \ c_e = c_e - f(e) \).
   - \( \forall e = (u,v), \text{if} \ f(e) > 0, \text{create} \ e' = (v,u) \) with \( c_{e'} = f(e) \)

\( G: \)
\( f: \)
\( G_f: \)

\( P = s \to a \to t \)
Ford-Fulkerson Algorithm

Max-Flow(G)

\[ f(e) = 0 \text{ for all } e \text{ in } G \]

while s-t path in \( G_f \) exists

\[ P = \text{simple s-t path in } G_f \]

\[ f' = \text{augment}(f, P) \]

\[ f = f' \]

\[ G_f = G_f' \]

return \( f \)

augment(f, P)

\[ b = \text{bottleneck}(P, f) \]

for each edge \((u, v)\) in \( P \)

if \((u, v)\) is a back edge

\[ f((v, u)) = b \]

else

\[ f((u, v)) = b \]

return \( f \)

Residual graph for flow \( f, G_f \):

- \( \forall e, \text{if } f(e) < c_e, \text{let } c_e = c_e - f(e) \).
- \( \forall e = (u, v), \text{if } f(e) > 0, \text{create } e' = (v, u) \text{ with } c_{e'} = f(e) \)

Residual graph for flow \( f, G_f \):

\[ P = s \rightarrow a \rightarrow b \rightarrow c \rightarrow t \]
**Ford-Fulkerson Algorithm**

Max-Flow(G)
- \( f(e) = 0 \) for all \( e \) in \( G \)
- while \( s-t \) path in \( G_f \) exists
  - \( P = \) simple \( s-t \) path in \( G_f \)
  - \( f' = \) augment \((f, P)\)
  - \( f = f' \)
  - \( G_f = G'_f \)
- return \( f \)

augment \((f, P)\)
- \( b = \) bottleneck \((P, f)\)
- for each edge \((u, v)\) in \( P \)
  - if \((u, v)\) is a back edge
    - \( f((v, u)) -= b \)
  - else
    - \( f((u, v)) += b \)
- return \( f \)

Residual graph for flow \( f, G_f \):
- \( \forall E, \text{if } f(e) < c_e, \text{let } c'_e = c_e - f(e). \)
- \( \forall E = (u, v), \text{if } f(e) > 0, \text{create } e' = (v, u) \text{ with } c'_e = f(e) \)
Ford-Fulkerson Algorithm

Max-Flow(G)

\[ f(e) = 0 \text{ for all } e \text{ in } G \]

while s-t path in \( G_f \) exists

\[ P = \text{simple s-t path in } G_f \]

\[ f' = \text{augment}(f, P) \]

\[ f = f' \]

\[ G_f = G_{f'} \]

return \( f \)

augment(f, P)

\[ b = \text{bottleneck}(P, f) \]

for each edge \((u, v)\) in \( P \)

if \((u, v)\) is a back edge

\[ f((v, u)) -= b \]

else

\[ f((u, v)) += b \]

return \( f \)

Residual graph for flow \( f, G_f \):  
- \( \forall e \), if \( f(e) < c_e \), let \( c_e = c_e - f(e) \).
- \( \forall e = (u, v) \), if \( f(e) > 0 \), create \( e' = (v, u) \) with \( c_{e'} = f(e) \)
Ford-Fulkerson Algorithm

Max-Flow(G)

\[ f(e) = 0 \text{ for all } e \text{ in } G \]

**while** s-t path in \( G_f \) exists

\[ P = \text{simple s-t path in } G_f \]

\[ f' = \text{augment}(f, P) \]

\[ f = f' \]

\[ G_f = G_{f'} \]

**return** \( f \)

Residual graph for flow \( f, G_f \):

- \( \forall e, f(e) < c_e, \text{let } c_e = c_e - f(e) \).
- \( \forall e = (u, v), \text{if } f(e) > 0, \text{create } e' = (v, u) \text{ with } c_{e'} = f(e) \)

**augment**(f, P)

\[ b = \text{bottleneck}(P, f) \]

**for** each edge \((u, v)\) in \( P \)

- **if** \((u, v)\) is a back edge

  \[ f((v, u)) -= b \]

- **else**

  \[ f((u, v)) += b \]

**return** \( f \)
Ford-Fulkerson Algorithm

Max-Flow(G)
  \( f(e) = 0 \) for all \( e \) in \( G \)
  \textbf{while} s-t path in \( G_f \) exists
    \( P = \) simple s-t path in \( G_f \)
    \( f' = \text{augment}(f, P) \)
    \( f = f' \)
  \( G_f = G_f' \)
  \textbf{return} \( f \)

augment(f, P)
  \( b = \text{bottleneck}(P,f) \)
  \textbf{for} each edge \((u, v)\) in \( P \)
    if \((u, v)\) is a back edge
      \( f((v, u)) -= b \)
    else
      \( f((u, v)) += b \)
  \textbf{return} \( f \)

Residual graph for flow \( f, G_f \):
- \( \forall e, \) if \( f(e) < c_e \), let \( c'_e = c_e - f(e) \).
- \( \forall e = (u, v), \) if \( f(e) > 0 \), create \( e' = (v, u) \) with \( c'_e = f(e) \).
Ford-Fulkerson Algorithm

Max-Flow(G)

1. \( f(e) = 0 \) for all \( e \) in \( G \)
2. while \( s \)-t path in \( G_f \) exists
   
   a. \( P = \) simple \( s \)-t path in \( G_f \)
   
   b. \( f' = \) augment\((f, P)\)
   
   c. \( f = f' \)
   
   d. \( G_f = G_f' \)

3. return \( f \)

augment\((f, P)\)

1. \( b = \) bottleneck\((P, f)\)
2. for each edge \((u, v)\) in \( P \)
   
   a. if \((u, v)\) is a back edge
      
      i. \( f((v, u)) = b \)
   
   b. else
      
      i. \( f((u, v)) = b \)

3. return \( f \)

Need to show:

1. Validity.
2. Running time.
3. Finds max flow.

Residual graph for flow \( f, G_f \):

- \( \forall e, \) if \( f(e) < c_e \), let \( c_e = c_e - f(e) \).
- \( \forall e = (u, v), \) if \( f(e) > 0 \), create \( e' = (v, u) \) with \( c_{e'} = f(e) \).