# Flow Networks
# CSCI 432

# Ford-Fulkerson Algorithm

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 $c_{e'} = f(e)$

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

Need to show:

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

# Validity

Theorem: If the edge capacities are integer, the algorithm will terminate.

Proof:

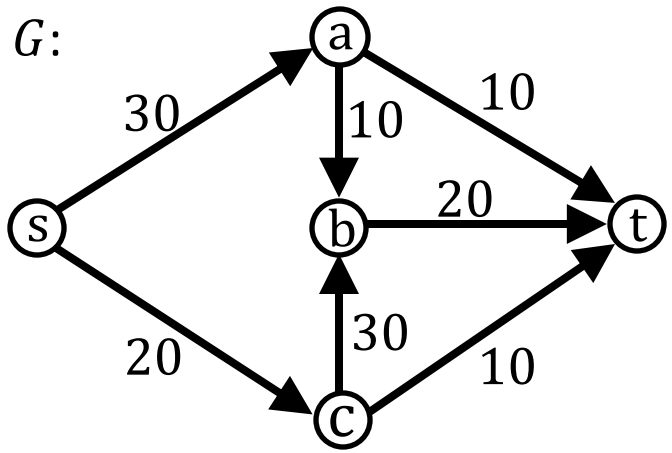
# Validity

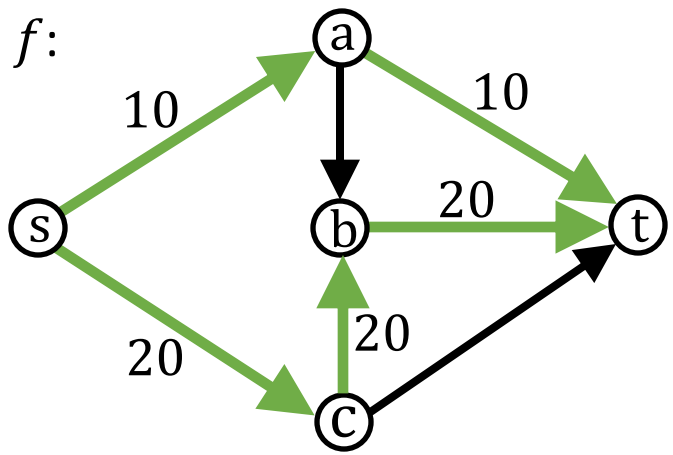
Theorem: If the edge capacities are integer, the algorithm will terminate.

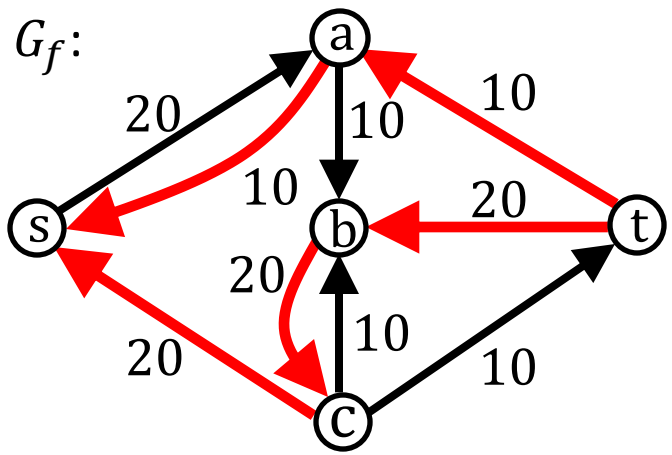
Proof:

- If the current flow is integer, the bottleneck (and the augmented flow) is also integer.

$$\text{bottleneck} = \min_{e \in \text{path}} \begin{cases} \text{capacity}_e - \text{flow}_e, & e \text{ is not a back edge} \\ \text{flow}_e, & e \text{ is a back edge} \end{cases}$$



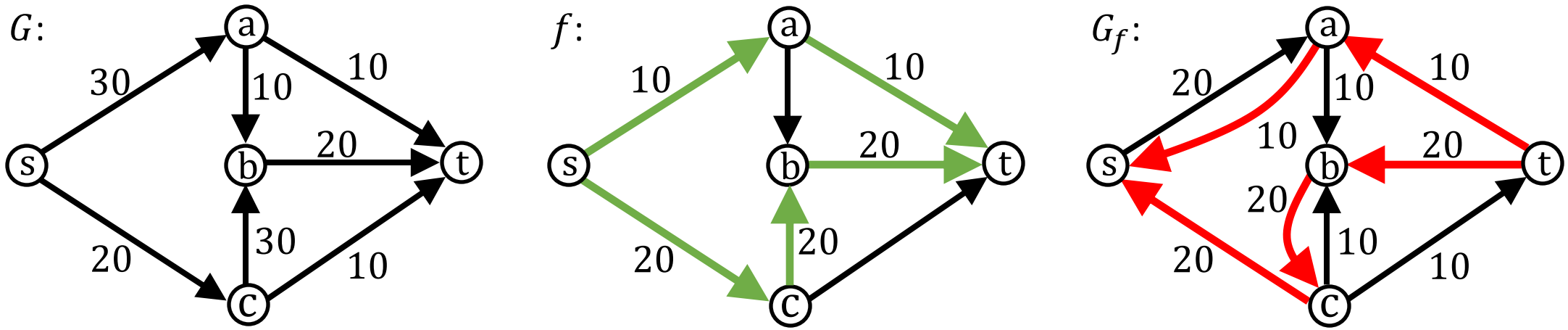
# Validity

Theorem: If the edge capacities are integer, the algorithm will terminate.

Proof:

- If the current flow is integer, the bottleneck (and the augmented flow) is also integer. Since the flow starts at 0 (integer) the flow values are always integer.

$$\text{bottleneck} = \min_{e \in path} \begin{cases} \text{capacity}_e - \text{flow}_e, & e \text{ is not a back edge} \\ \text{flow}_e, & e \text{ is a back edge} \end{cases}$$



# Validity

Theorem: If the edge capacities are integer, the algorithm will terminate.

Proof:

- If the current flow is integer, the bottleneck (and the augmented flow) is also integer. Since the flow starts at 0 (integer) the flow values are always integer.
- The flow value always increases by at least 1 each iteration (flow value must increase and bottleneck is an integer > 0).

# Validity

Theorem: If the edge capacities are integer, the algorithm will terminate.

Proof:

- If the current flow is integer, the bottleneck (and the augmented flow) is also integer. Since the flow starts at 0 (integer) the flow values are always integer.
- The flow value always increases by at least 1 each iteration (flow value must increase and bottleneck is an integer > 0).
- The Max Flow is finite. (e.g., Bounded by sum of capacities into $t$.)

# Validity

Theorem: If the edge capacities are integer, the algorithm will terminate.

Proof:

- If the current flow is integer, the bottleneck (and the augmented flow) is also integer. Since the flow starts at 0 (integer) the flow values are always integer.
- The flow value always increases by at least 1 each iteration (flow value must increase and bottleneck is an integer > 0).
- The Max Flow is finite. (e.g., Bounded by sum of capacities into $t$.)

Thus, the algorithm will go through at most |Max Flow| iterations.

# Validity

Theorem: If the edge capacities are integer, the algorithm will terminate.

Proof:

- If the current flow is integer, the bottleneck (and the augmented flow) is also integer. Since the flow starts at 0 (integer) the flow values are always integer.
- The flow value always increases by at least 1 each iteration (flow value must increase and bottleneck is an integer > 0).
- The Max Flow is finite. (e.g., Bounded by sum of capacities into $t$.)

Thus, the algorithm will go through at most |Max Flow| iterations.

**Note: This does not hold for general edge capacities (irrational edge capacities can lead to non-terminating scenarios).**

# Ford-Fulkerson Algorithm

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 $c_{e'} = f(e)$

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

Need to show:

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

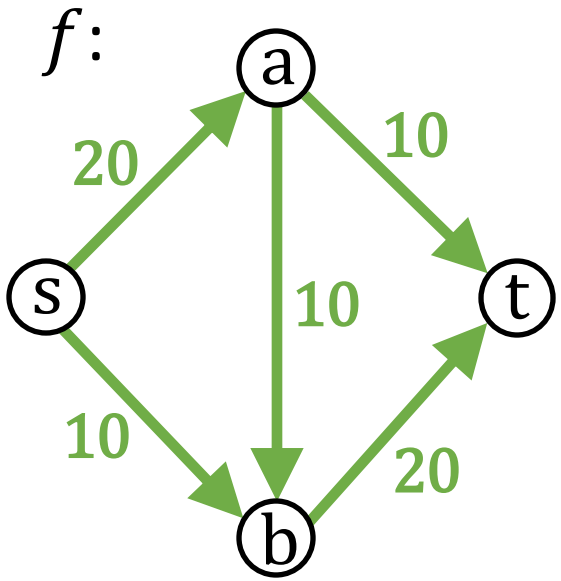
# Flows in Residual Graphs

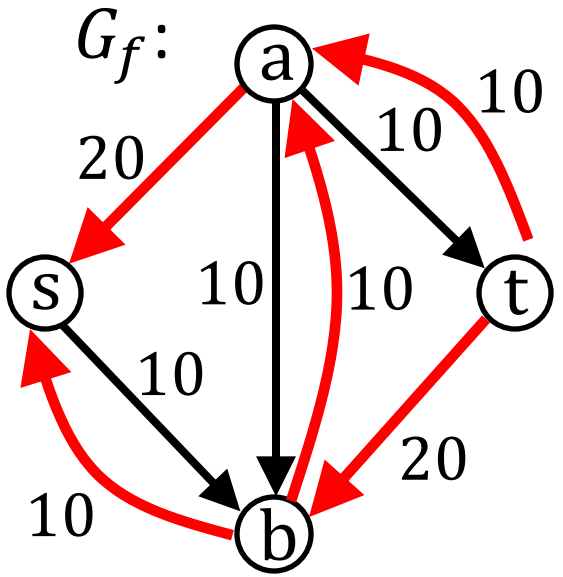
Let $P$ be a simple $s - t$ path in $G_f$.

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

```
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
```

Claim: `augment(f, P)` is a flow in $G$.




$P = s \to b \to a \to t$

# Validity

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$: ?

```
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
```

# Validity

```
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
```

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Capacity constraint ($f'(e) \leq c_e$)

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

```
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
```

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Capacity constraint ($f'(e) \leq c_e$)

If $e$ is a regular (forward) edge,

$$f'(e) = f(e) + \text{bottleneck}(P, f)$$

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

```
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
```

Theorem: f' = augment(f, P) is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Capacity constraint ($f'(e) \leq c_e$)

If $e$ is a regular (forward) edge,

$$f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + \boxed{c_e - f(e)}$$

residual capacity

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

```
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
```

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Capacity constraint ($f'(e) \leq c_e$)

If $e$ is a regular (forward) edge,

$$f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + c_e - f(e)$$

because bottleneck is min over all $e$.

residual capacity

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

```
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
```

Theorem: f' = augment(f, P) is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Capacity constraint ($f'(e) \leq c_e$)

If $e$ is a regular (forward) edge,

$$f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + \boxed{c_e - f(e)} = c_e$$

because bottleneck is min over all $e$.

residual capacity

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

```
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
```

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Capacity constraint ($f'(e) \leq c_e$)

If $e$ is a regular (forward) edge,

$$f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + c_e - f(e) = c_e$$

If $e$ is a back edge, let $e'$ be the forward edge,

$$f'(e') = f(e') - \text{bottleneck}(P, f)$$

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

```
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
```

Theorem: f' = augment(f, P) is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Capacity constraint ($f'(e) \leq c_e$)

If $e$ is a regular (forward) edge,

$$f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + c_e - f(e) = c_e$$

If $e$ is a back edge, let $e'$ be the forward edge,

$$f'(e') = f(e') - \text{bottleneck}(P, f) \leq f(e')$$

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

```
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
```

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Capacity constraint ($f'(e) \leq c_e$)

If $e$ is a regular (forward) edge,

$$f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + c_e - f(e) = c_e$$

If $e$ is a back edge, let $e'$ be the forward edge,

$$f'(e') = f(e') - \text{bottleneck}(P, f) \leq f(e') \leq c_{e'}$$

since $f$ was a valid flow

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

$v \notin P$ **not affected.**

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

# Validity

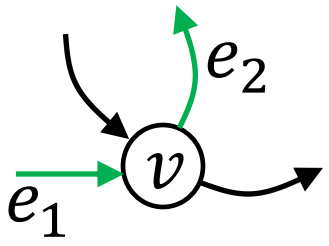
Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

?



# Validity

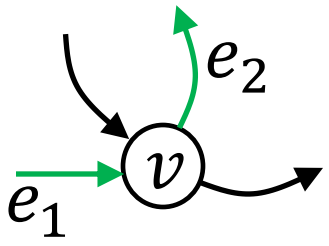
Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

input′($v$) = input($v$) + bottleneck($P, f$) **only edge changed is $e_1$.**



# Validity

Theorem: `f' = augment(f, P)` is a flow in G.

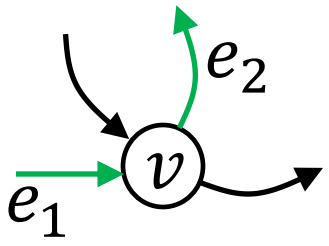
Proof: Need to verify, for each edge/node in $P$:

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

input′($v$) = input($v$) + bottleneck($P, f$)

output′($v$) = output($v$) + bottleneck($P, f$) **only edge changed is $e_2$.**



# Validity

Theorem: `f' = augment(f, P)` is a flow in G.

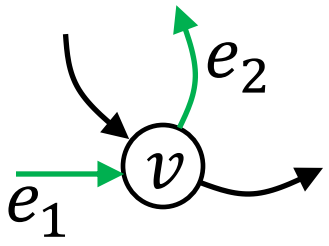
Proof: Need to verify, for each edge/node in $P$:

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

input′($v$) = input($v$) + bottleneck($P, f$)
output′($v$) = output($v$) + bottleneck($P, f$)
$\Rightarrow$ input′($v$) = output′($v$)

since input($v$) = output($v$)



# Validity

```
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
```

Theorem: `f' = augment(f, P)` is a flow in G.

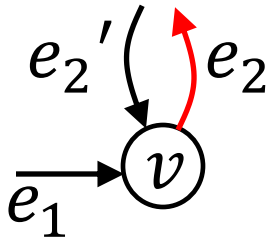
Proof: Need to verify, for each edge/node in $P$:

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

input′($v$) = input($v$) + bottleneck($P, f$)
output′($v$) = output($v$) + bottleneck($P, f$)
$\Rightarrow$ input′($v$) = output′($v$)

Case 2: Input $e_1$ is regular edge, output $e_2$ is back edge.




?

# Validity

```
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
```

Theorem: f' = augment(f, P) is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

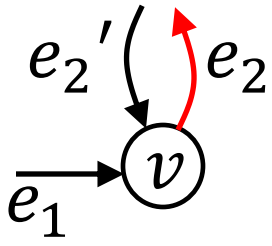
Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

input′($v$) = input($v$) + bottleneck($P, f$)
output′($v$) = output($v$) + bottleneck($P, f$)
$\Rightarrow$ input′($v$) = output′($v$)

Case 2: Input $e_1$ is regular edge, output $e_2$ is back edge.

input′($v$) = input($v$) + bottleneck($P, f$) - bottleneck($P, f$)




extra on $e_1$

extra on $e_2$ = subtracted from $e_2'$

# Validity

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

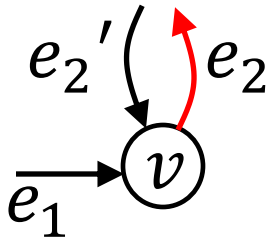
Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

input′($v$) = input($v$) + bottleneck($P, f$)
output′($v$) = output($v$) + bottleneck($P, f$)
} $\Rightarrow$ input′($v$) = output′($v$)

Case 2: Input $e_1$ is regular edge, output $e_2$ is back edge.

input′($v$) = input($v$)
output′($v$) = output($v$)



# Validity

Theorem: `f' = augment(f, P)` is a flow in G.

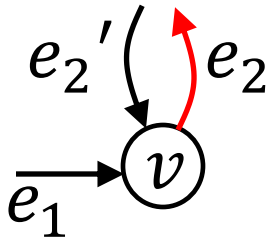
Proof: Need to verify, for each edge/node in $P$:

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

input′($v$) = input($v$) + bottleneck($P, f$)
output′($v$) = output($v$) + bottleneck($P, f$)
$\Rightarrow$ input′($v$) = output′($v$)

Case 2: Input $e_1$ is regular edge, output $e_2$ is back edge.




input′($v$) = input($v$)
output′($v$) = output($v$)
$\Rightarrow$ input′($v$) = output′($v$)

since input($v$) = output($v$)

# Validity

Theorem: `f' = augment(f, P)` is a flow in G.

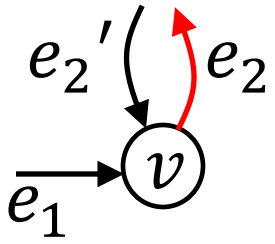
Proof: Need to verify, for each edge/node in $P$:

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

input′($v$) = input($v$) + bottleneck($P, f$)
output′($v$) = output($v$) + bottleneck($P, f$)
$\Rightarrow$ input′($v$) = output′($v$)

Case 2: Input $e_1$ is regular edge, output $e_2$ is back edge.




input′($v$) = input($v$)
output′($v$) = output($v$)
$\Rightarrow$ input′($v$) = output′($v$)
since input($v$) = output($v$)

Case 3 & 4: Same arguments…

# Validity

Theorem: `f' = augment(f, P)` is a flow in G.

Proof: Need to verify, for each edge/node in $P$:

Conservation of flow constraint (input($v$) = output($v$), $\forall$ internal $v \in P$)

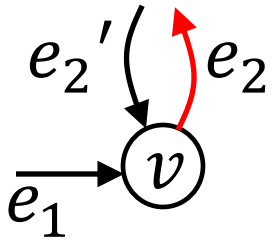
Case 1: Input $e_1$ is regular edge, output $e_2$ is regular edge.

$$\left.\begin{aligned} \text{input}'(v) &= \text{input}(v) + \text{bottleneck}(P, f) \\ \text{output}'(v) &= \text{output}(v) + \text{bottleneck}(P, f) \end{aligned}\right\} \Rightarrow \text{input}'(v) = \text{output}'(v)$$

Case 2: Input $e_1$ is regular edge, output $e_2$ is back edge.

$$\left.\begin{aligned} \text{input}'(v) &= \text{input}(v) \\ \text{output}'(v) &= \text{output}(v) \end{aligned}\right\} \Rightarrow \text{input}'(v) = \text{output}'(v)$$

since input($v$) = output($v$)




Case 3 & 4: Same arguments…

**So, `f' = augment(f, P)` will be a valid flow!**

# Validity

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

Need to show:

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

# Running Time

Assuming integer edge capacities:

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

# Running Time

Assuming integer edge capacities:

While loop runs at most ???

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

# Running Time

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.

**Flow starts at 0, increases by at least 1 each iteration.**

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

# Running Time

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.

Find $s - t$ path…?

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

# Running Time

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.
Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$

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

# Running Time

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.
Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$
`augment(f, P)`: ???

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

# Running Time

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.
Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$
`augment(f, P)` just traverses edges: $O(|E|)$

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

# Running Time

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 $c_{e'} = f(e)$

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.

Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$

`augment(f, P)` just traverses edges: $O(|E|)$

Update $G_f$: ???

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

# Running Time

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 $c_{e'} = f(e)$

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.
Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$
`augment(f, P)` just traverses edges: $O(|E|)$
Update $G_f$ (for each $e \in G$, make $e$ and $e' \in G_f$): $O(|E|)$

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

# Running Time

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.
Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$
`augment(f, P)` just traverses edges: $O(|E|)$
Update $G_f$ (for each $e \in G$, make $e$ and $e' \in G_f$): $O(|E|)$

Total = $O(|f_{OPT}|\ (3|E| + |V|)) = O(|E| \cdot |f_{OPT}|)$

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

# Running Time

Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.
Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$
`augment(f, P)` just traverses edges: $O(|E|)$
Update $G_f$ (for each $e \in G$, make $e$ and $e' \in G_f$): $O(|E|)$

Total = $O(|f_{OPT}|\ (3|E| + |V|)) = O(|E| \cdot |f_{OPT}|)$

**Good**

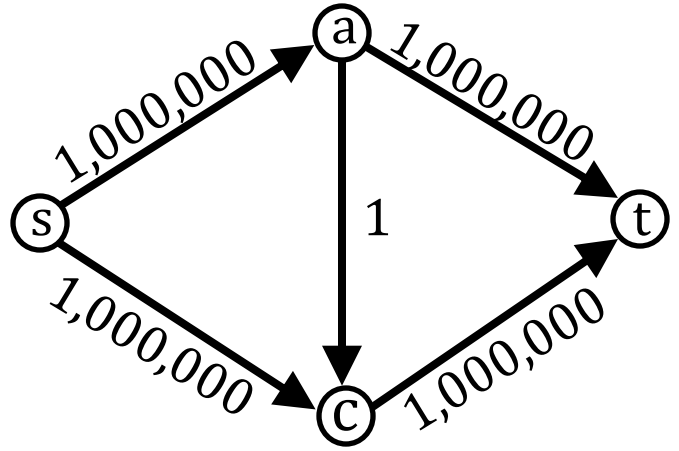
**or**

**Bad?**

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

# Running Time




Assuming integer edge capacities:

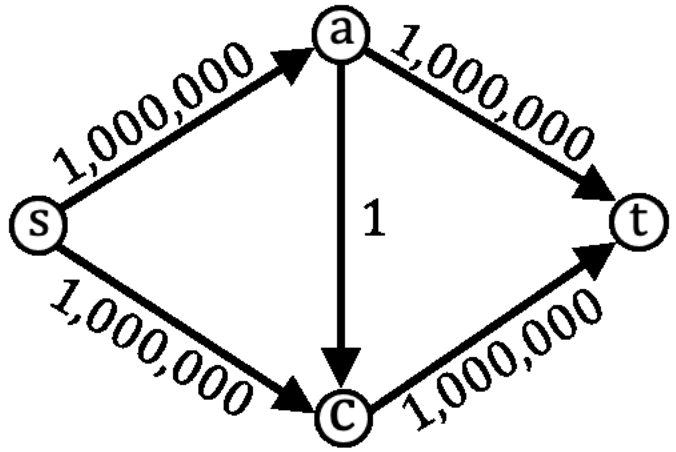
While loop runs at most $|f_{OPT}|$ times.
Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$
`augment(f, P)` just traverses edges: $O(|E|)$
Update $G_f$ (for each $e \in G$, make $e$ and $e' \in G_f$): $O(|E|)$

Total = $O(|f_{OPT}|\ (3|E| + |V|)) = O(|E| \cdot |f_{OPT}|)$

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

# Running Time




Assuming integer edge capacities:

While loop runs at most $|f_{OPT}|$ times.
Find $s - t$ path (BFS/DFS): $O(|E| + |V|)$
`augment(f, P)` just traverses edges: $O(|E|)$
Update $G_f$ (for each $e \in G$, make $e$ and $e' \in G_f$): $O(|E|)$

Total = $O(|f_{OPT}|\ (3|E| + |V|)) = O(|E| \cdot |f_{OPT}|)$

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

# Ford-Fulkerson Algorithm

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

# Edmonds-Karp Algorithm

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = shortest s-t path in Gf
      f' = augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

**Shortest, as in smallest number of edges.**

```
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
```

# Edmonds-Karp Algorithm

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = shortest s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   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
```

Running Time: $O(|V||E|^2)$

# Edmonds-Karp Algorithm

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = shortest s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   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
```

Running Time: $O(|V||E|^2)$

improved to $O(|V||E|)$ [Orlin, 2013]

# Edmonds-Karp Algorithm

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = shortest s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   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
```

Running Time: $O(|V||E|^2)$

improved to $O(|V||E|)$ [Orlin, 2013]

improved to $O\left(|E|^{1+o(1)}\right)$ [Chen et al., 2022]

# Edmonds-Karp Algorithm

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = shortest s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   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
```

Running Time: $O(|V||E|^2)$

improved to $O(|V||E|)$ [Orlin, 2013]

improved to $O\left(|E|^{1+o(1)}\right)$ [Chen et al., 2022]

$\in \boldsymbol{O}(\textbf{Max Flow})$

# Optimality

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

Need to show:

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

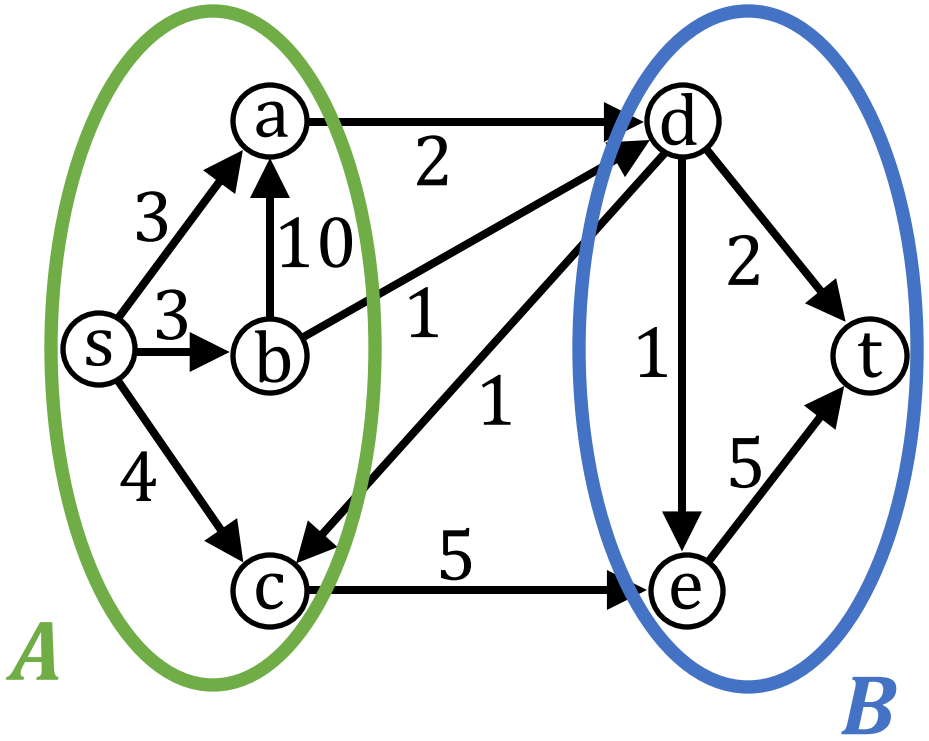
# Optimality

Theorem: The flow returned by the Ford-Fulkerson algorithm is a maximum flow.

Proof: ...

# Optimality

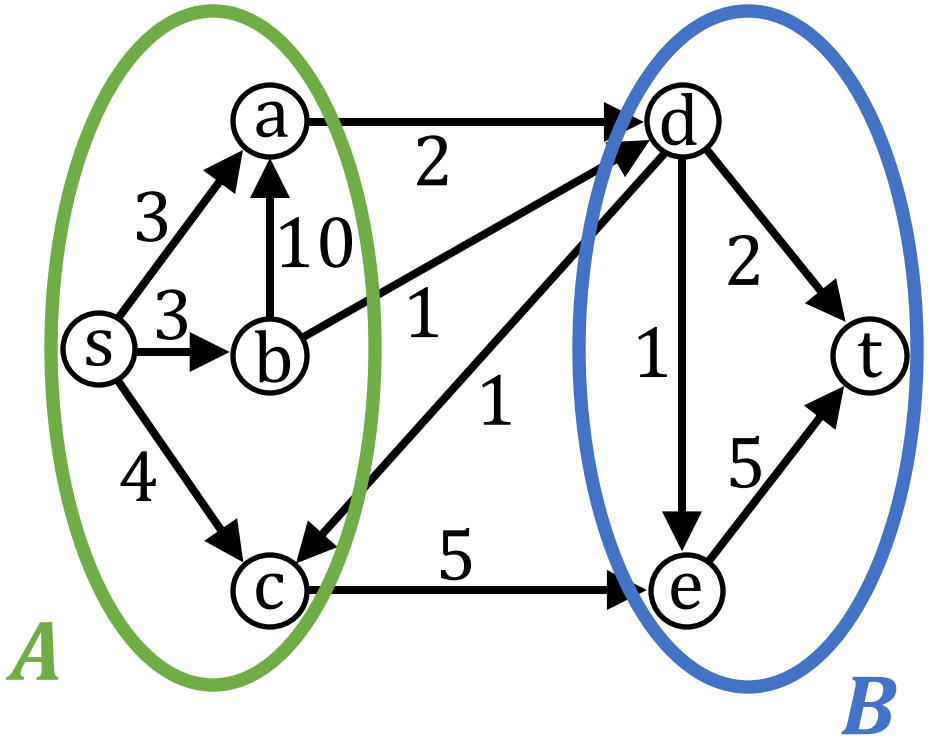
Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.




$$c(A, B) = 8$$

# Optimality

Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.

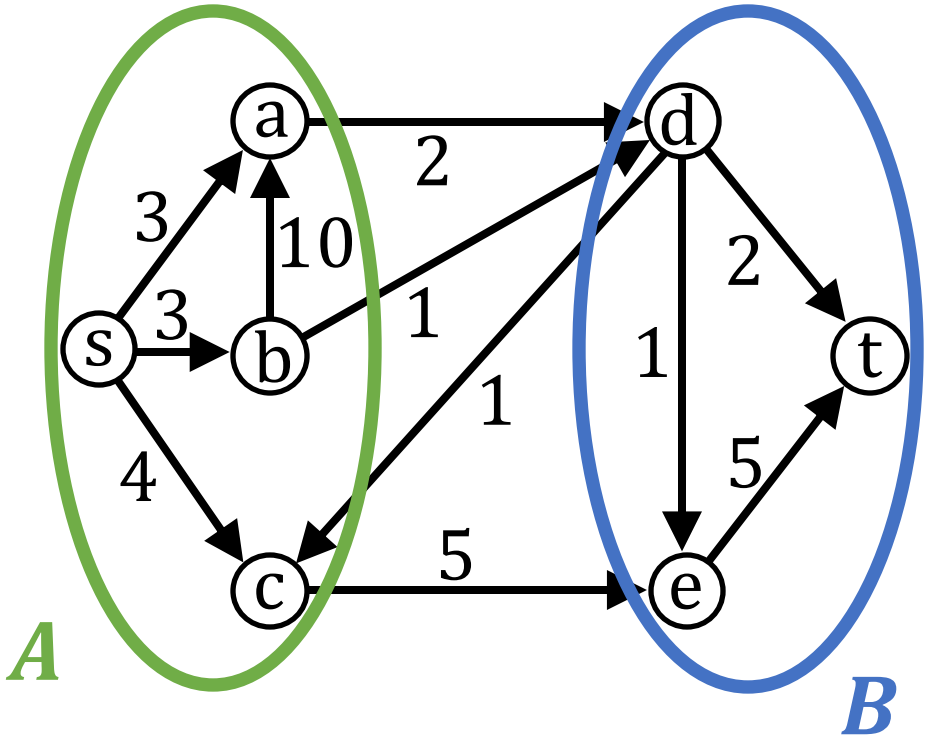



$$c(A, B) = 8$$

Game Plan:

# Optimality

Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.



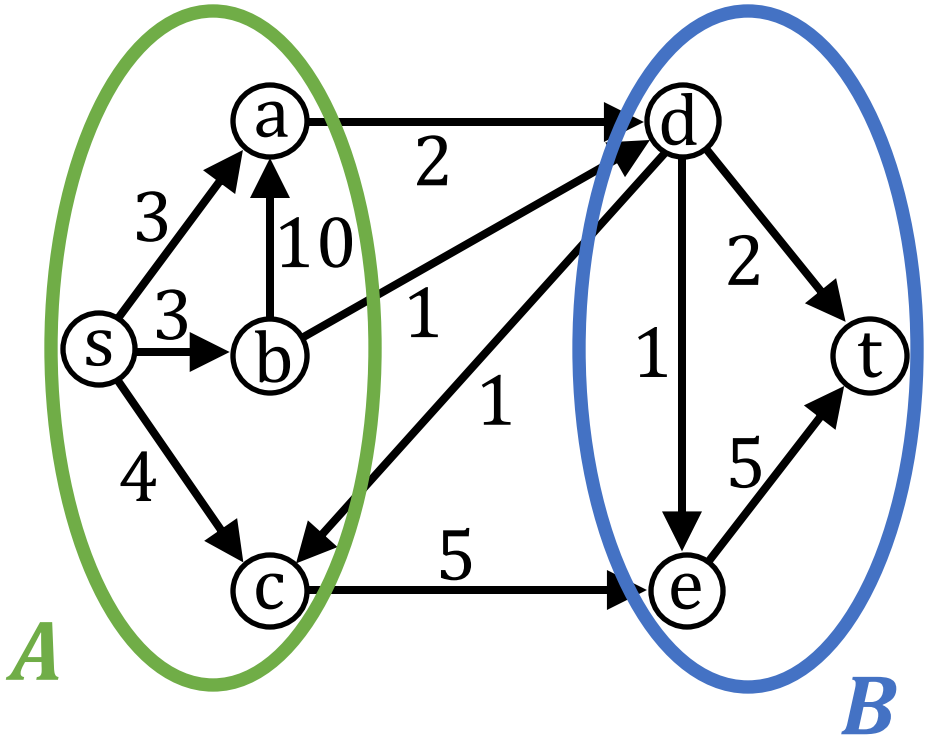

$$c(A, B) = 8$$

Game Plan:

1. Show that value of every flow is $\leq$ capacity of every cut.

# Optimality

Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.



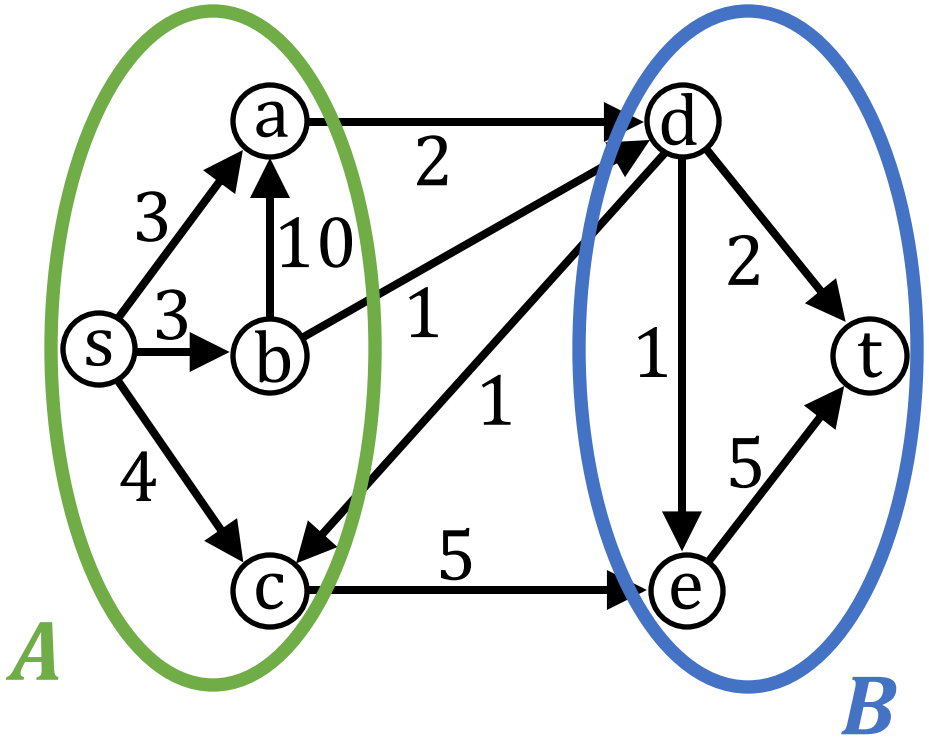

$$c(A, B) = 8$$

Game Plan:

1. Show that value of every flow is $\leq$ capacity of every cut.

# Optimality

Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.



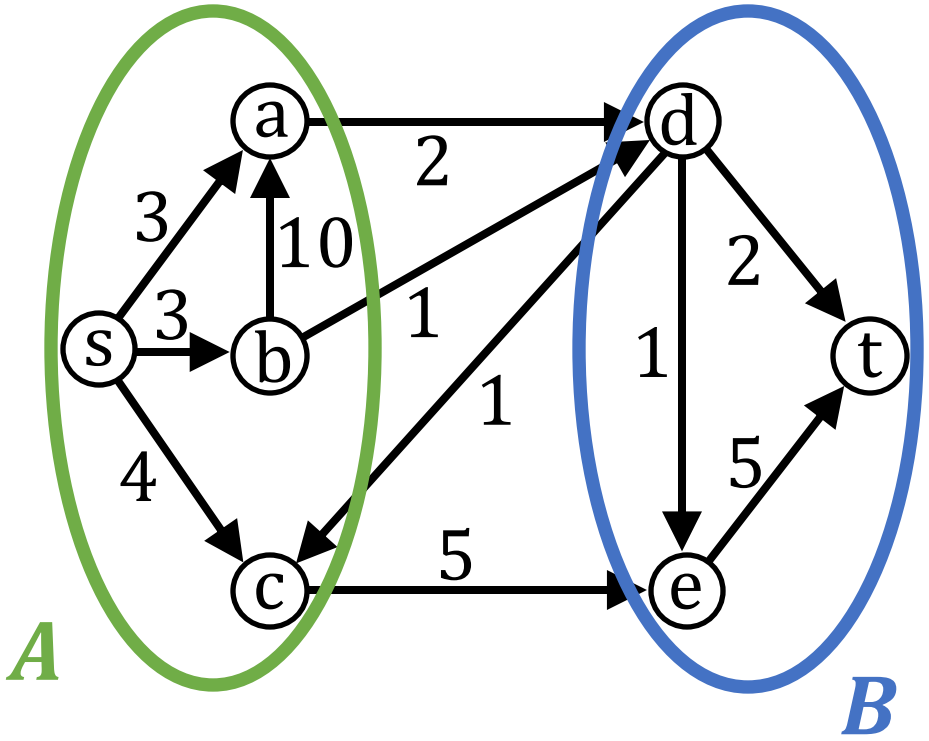

$$c(A, B) = 8$$

Game Plan:

1. Show that value of every flow is $\leq$ capacity of every cut.

# Optimality

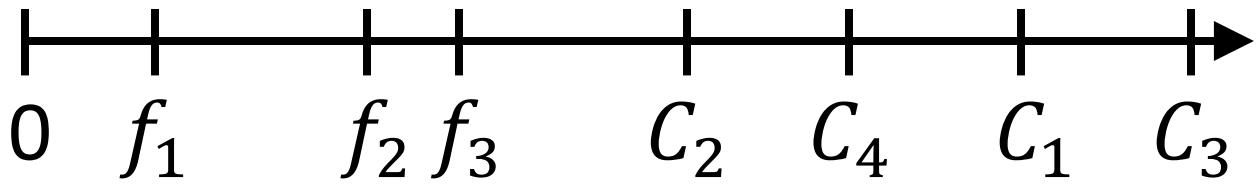
Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.




$$c(A, B) = 8$$

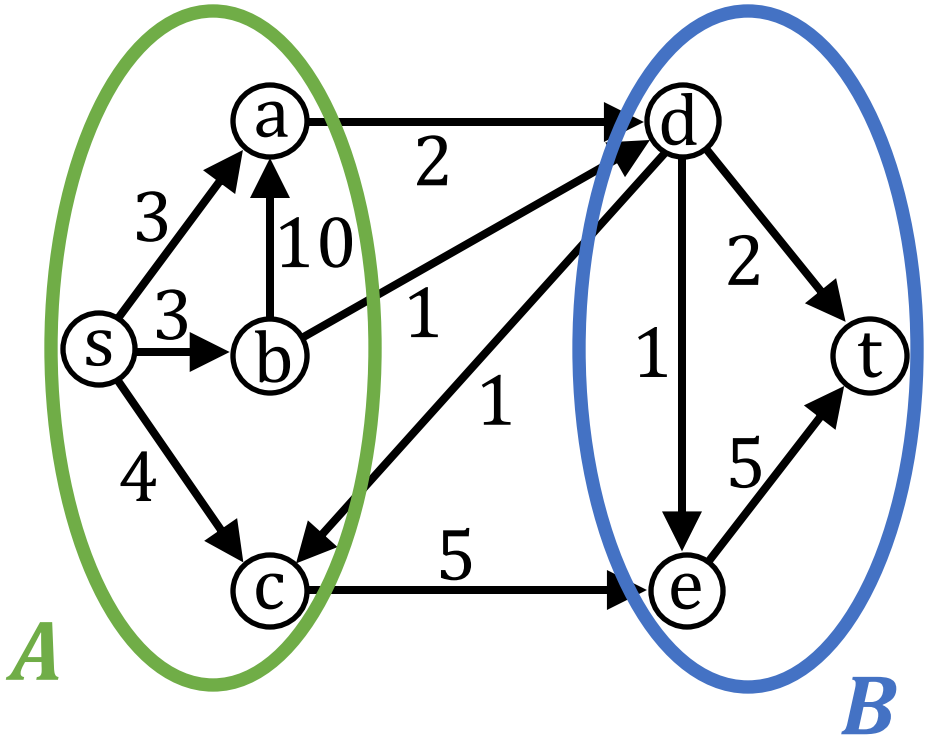
Game Plan:

1. Show that value of every flow is $\leq$ capacity of every cut.



# Optimality

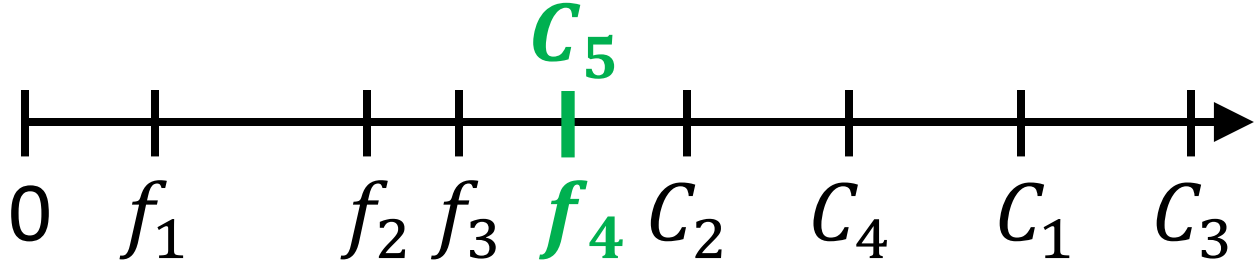
Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.




$$c(A, B) = 8$$

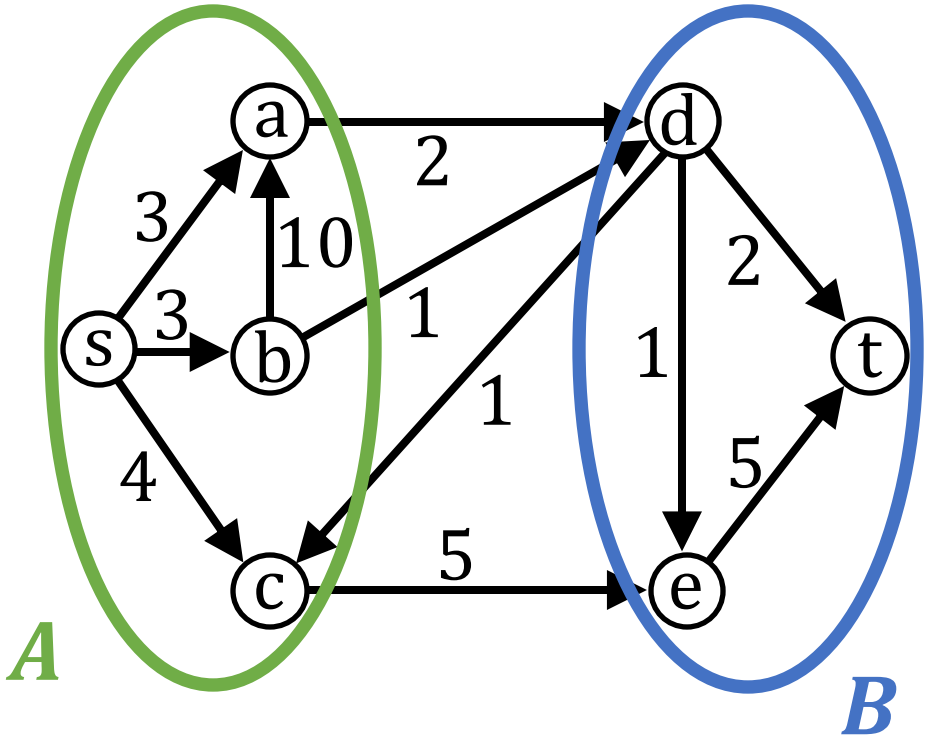
Game Plan:

1. Show that value of every flow is $\leq$ capacity of every cut.



# Optimality

Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.




$$c(A, B) = 8$$

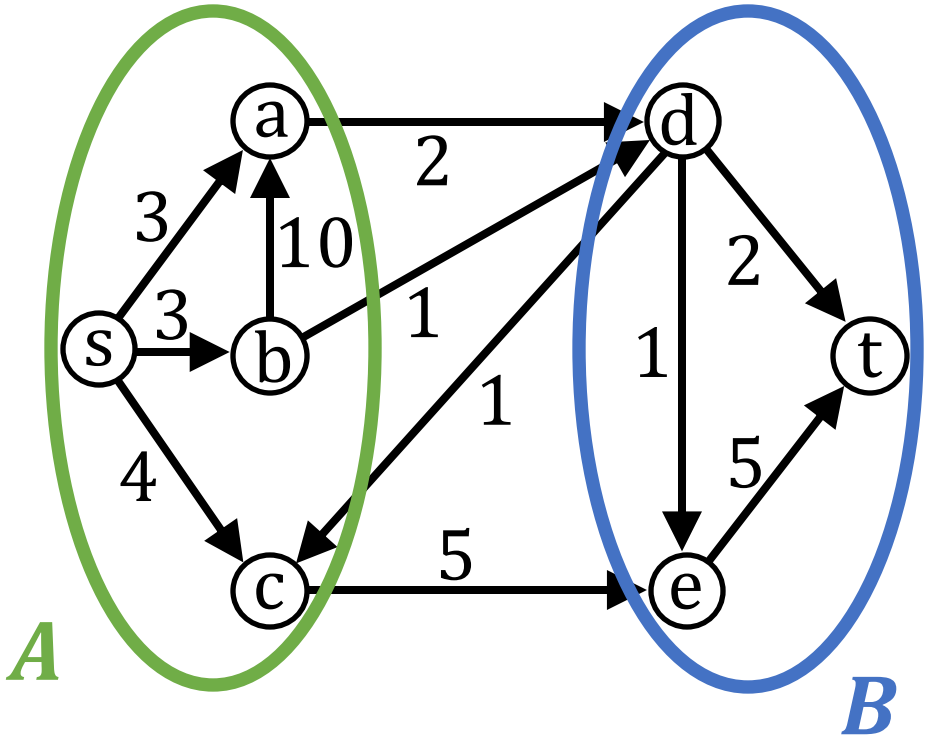
Game Plan:

1. Show that value of every flow is $\leq$ capacity of every cut.

**Consequence: If we find some flow whose value equals the capacity of some cut, it must be the optimal flow.**

# Optimality

Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.



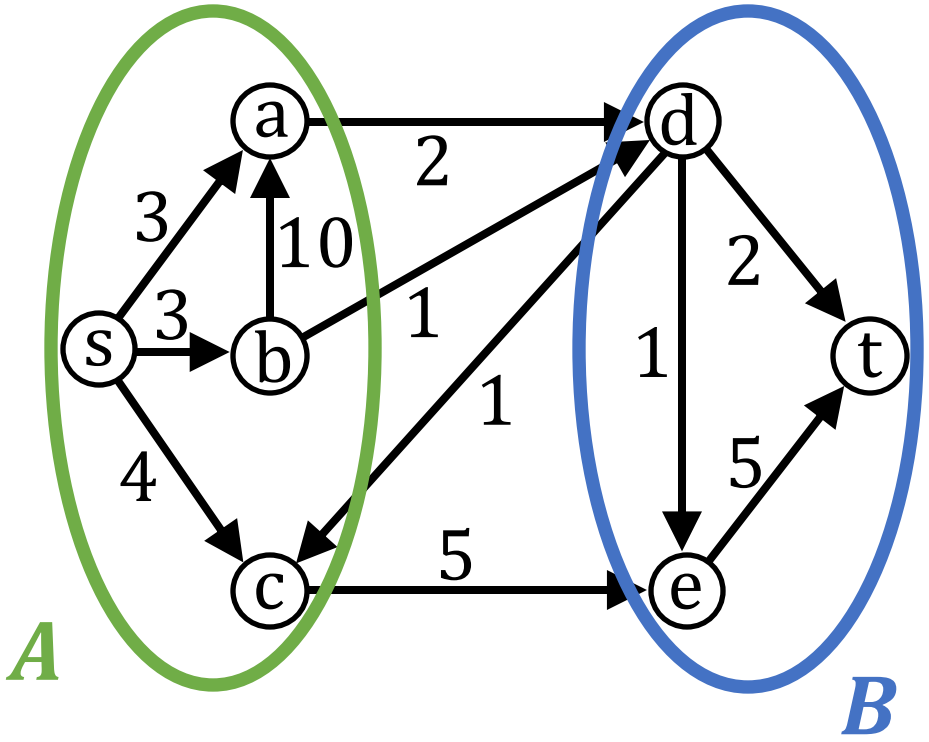

$$c(A, B) = 8$$

Game Plan:

1. Show that value of every flow is $\leq$ capacity of every cut.
2. Given a flow where there are no s-t paths left in the residual graph, there is a specific cut whose capacity = flow value.

# Optimality

Definitions: Suppose $G$ is a flow network and nodes in $G$ are divided into two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $s \in A$ and $t \in B$. We call $(A, B)$ an $s - t$ *cut*. The *capacity* of the cut, $c(A, B)$, is the sum of capacities of all edges out of A.




$$c(A, B) = 8$$

Game Plan:

1. Show that value of every flow is $\leq$ capacity of every cut.
2. Given a flow where there are no s-t paths left in the residual graph, there is a specific cut whose capacity = flow value.

**Consequence: The algorithm is optimal**