Flow Networks
CSCI 532

Test 1 Logistics

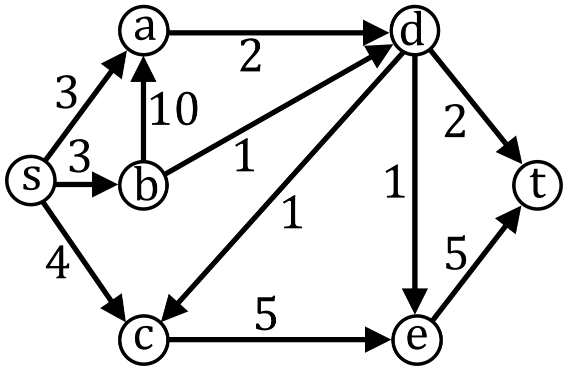
1. During class on Thursday 9/18.
2. You can bring your book and any notes you would like, but no electronic devices.
3. You may assume anything proven in class or on homework.
4. Three questions (12 points):
   1) Prove something related to MSTs (5 points).
   2) Identify recursive optimal substructure function for graph problem (5 points).
   3) Prove/disprove flow network property (2 points).

Flow Network

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

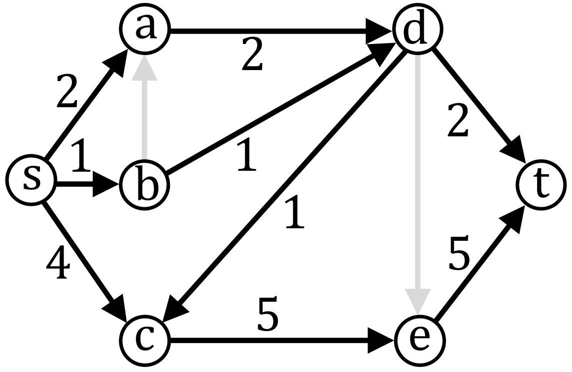
Flow Network:

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



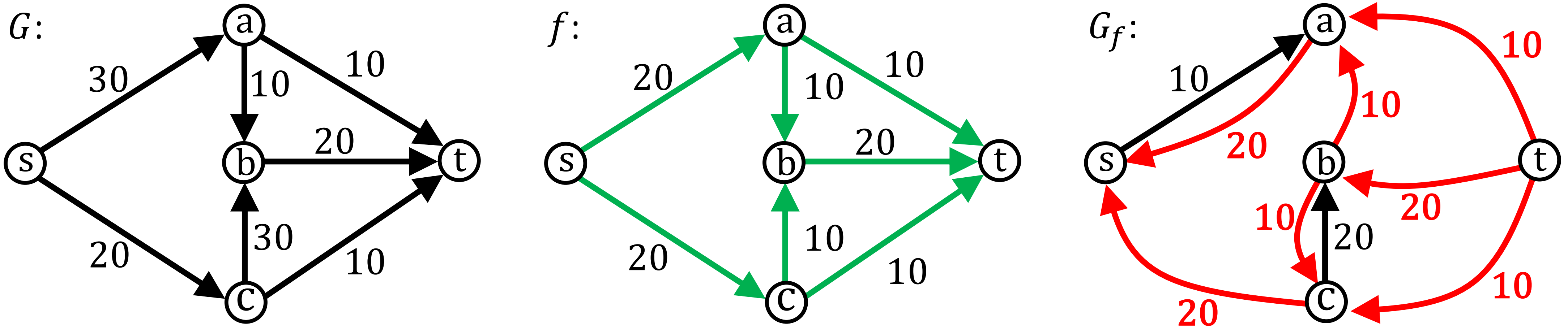
An $s - t$ flow is a function $f\colon E \;\rightarrow \mathbb{R}^+$ such that:

- $0 \le f(e) \le c_e, \forall e \in E$. (capacity constraint)
- $\sum_{e \epsilon \text{input}(v)} f(e) = \sum_{e \in \text{output}(v)} f(e)\,, \forall v \in V \setminus \{s, t\}$.
  (conservation of flow constraint)
- Value of flow = $val(f)$ = $\sum_{e \epsilon \text{output}(s)} f(e)$ = $\sum_{e \epsilon \text{input}(t)} f(e)$



Total flow = 7

Ford-Fulkerson




```
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
```

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

2. If edge capacities are integer-valued, the algorithm will terminate.

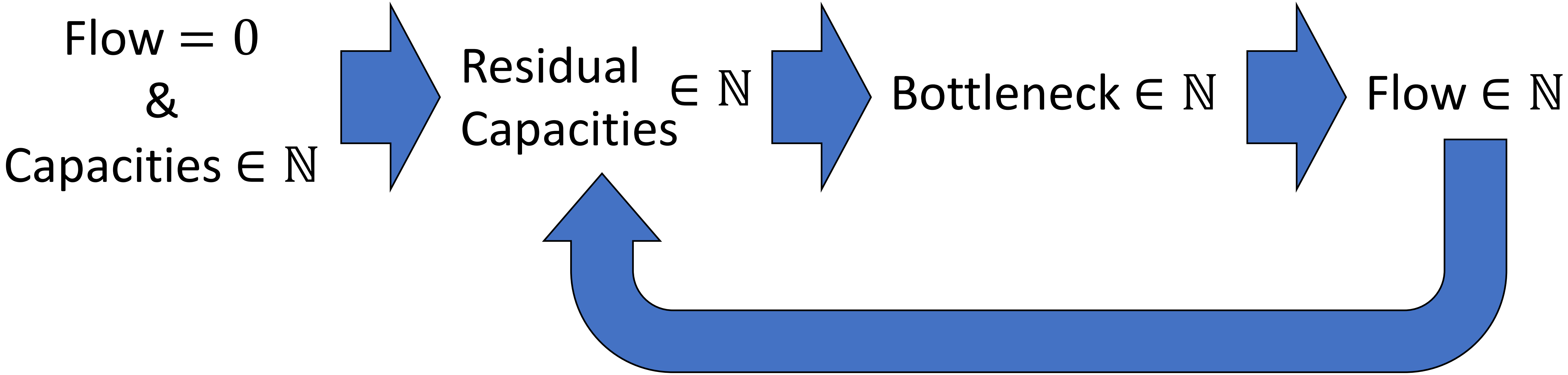
3. If an iteration starts with a valid flow, it ends with a valid flow.

4. The first iteration starts with a valid flow.

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.



Ford-Fulkerson

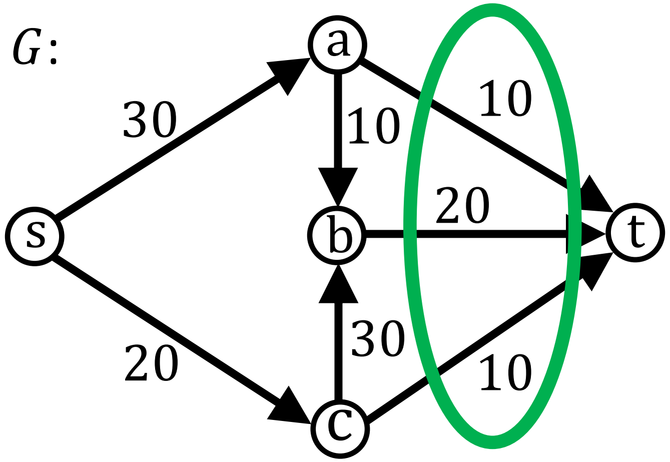
Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

    Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

    With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.



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

Ford-Fulkerson

Claims:

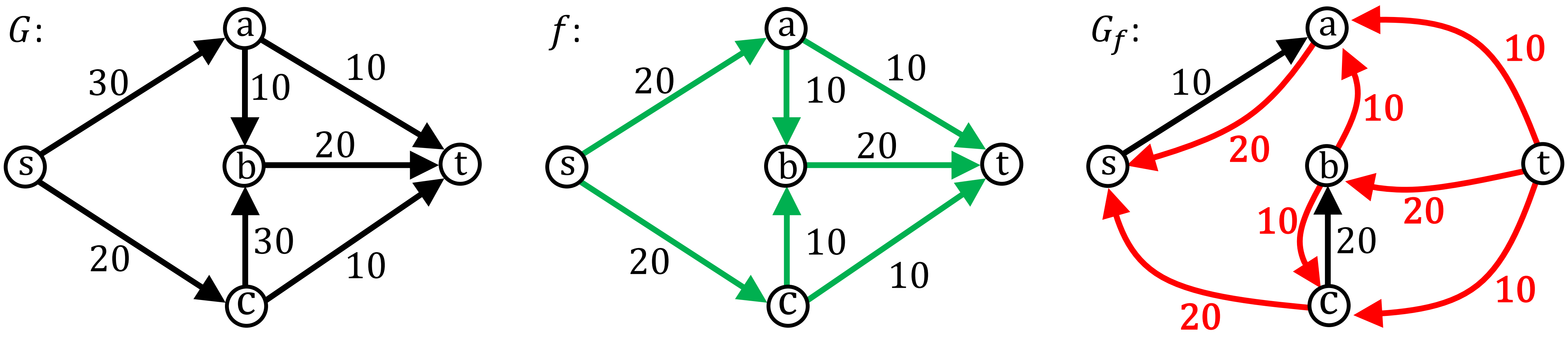
1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Ford-Fulkerson

If an iteration starts with a valid flow, it ends with a valid flow.




```
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
```

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

We only need to consider nodes/edges on the path. Other nodes/edges aren't modified.

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

We only need to consider nodes/edges on the path. Other nodes/edges aren't modified.

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

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Can't overflow capacities on back edges, because…

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

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq |\text{Max Flow}|$ iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Can't overflow capacities on back edges, because we are removing flow from them. So, if they were valid, less flow doesn't change that.

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

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Can't overflow capacities on forward edges since...

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

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Can't overflow capacities on forward edges since bottleneck = $\min_e(c_e - f_e)$

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

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Can't overflow capacities on forward edges since bottleneck = $\min_e(c_e - f_e)$

Thus, $f'_e = f_e +$ bottleneck

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

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Can't overflow capacities on forward edges since bottleneck = $\min_e(c_e - f_e)$

Thus, $f'_e = f_e +$ bottleneck

$$\leq f_e + c_e - f_e = c_e$$

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

Ford-Fulkerson

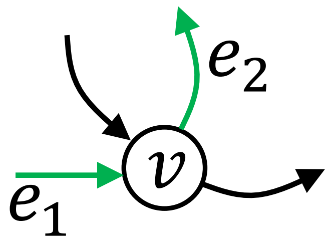
Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a forward edge.



Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

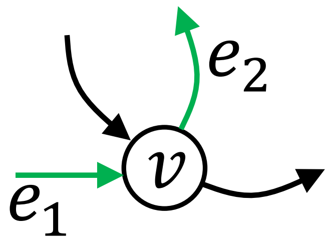
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a forward edge.



Then, $f'_{in} =$

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

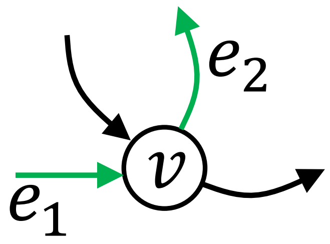
2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a forward edge.

Then, $f'_{in} = f_{in} +$ bottleneck



Ford-Fulkerson

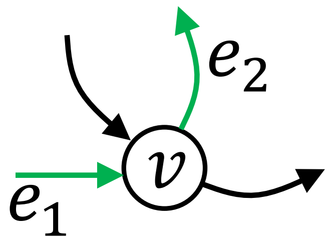
Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a forward edge.



Then, $f'_{in} = f_{in} +$ **bottleneck** $= f_{out} +$ **bottleneck**

Ford-Fulkerson

Claims:

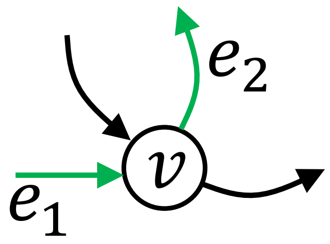
1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a forward edge.

Then, $f'_{in} = f_{in} + \text{bottleneck} = f_{out} + \text{bottleneck} = f'_{out}$



Ford-Fulkerson

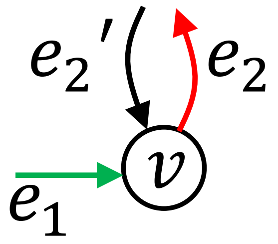
Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a back edge.



Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

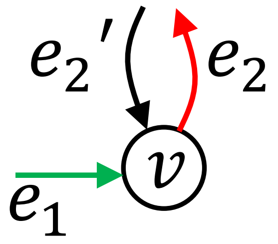
2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a back edge.

Then, $f'_{in} =$



Ford-Fulkerson

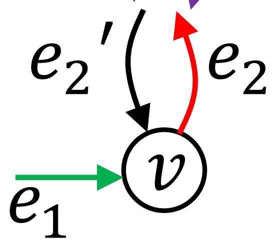
Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a v

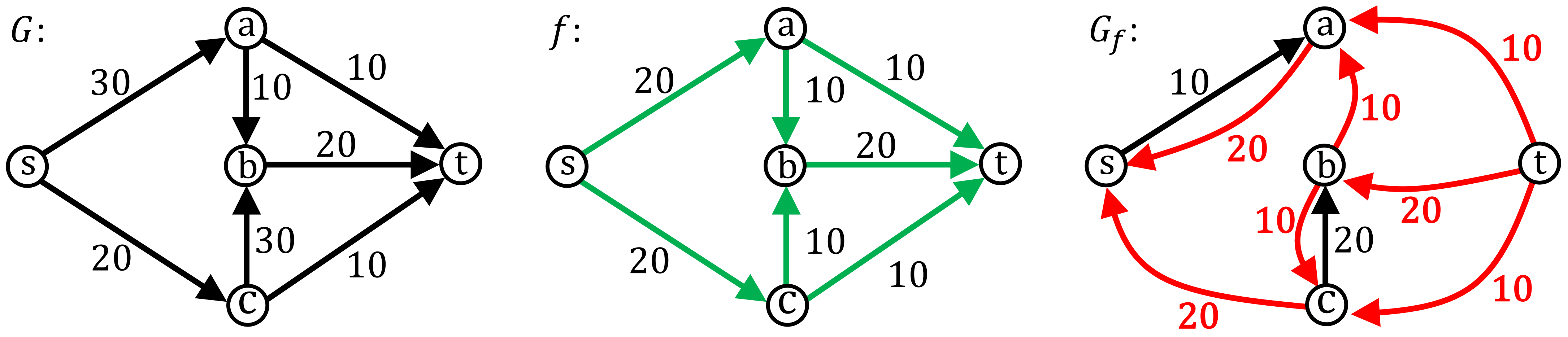
Flow on back edge removes from forward edge.



Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a back edge.

Then, $f'_{in} = f_{in} +$ bottleneck

Ford-Fulkerson




```
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

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

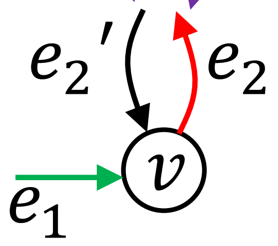
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a v

Flow on back edge removes from forward edge.



Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a back edge.

Then, $f'_{in} = f_{in} +$ bottleneck

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

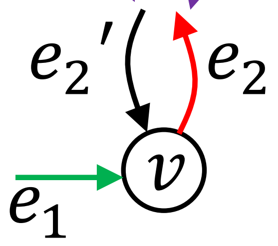
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a

Flow on back edge removes from forward edge.



Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a back edge.

Then, $f'_{in} = f_{in} +$ bottleneck – bottleneck

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

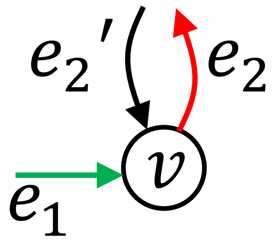
2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a forward edge and the edge out is a back edge.

Then, $f'_{in} = f_{in} + \textbf{bottleneck} - \textbf{bottleneck} = f_{in} = f_{out} = f'_{out}$



Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.
2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.
3. If an iteration starts with a valid flow, it ends with a valid flow.

Considering a node on the path. Suppose the edge in is a back edge and the edge out is a forward edge.

Considering a node on the path. Suppose the edge in is a back edge and the edge out is a back edge.

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

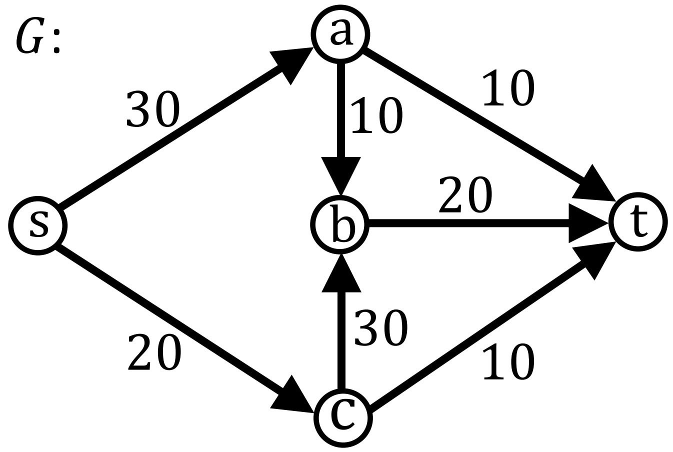
3. If an iteration starts with a valid flow, it ends with a valid flow.
   Each iteration sends flow along a residual $s - t$ path without violating capacities or conservation of flow. So, the resulting flow is valid.

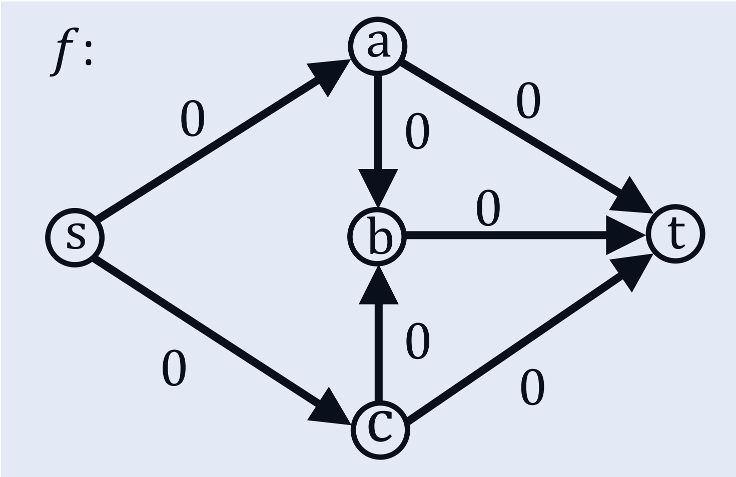
Ford-Fulkerson

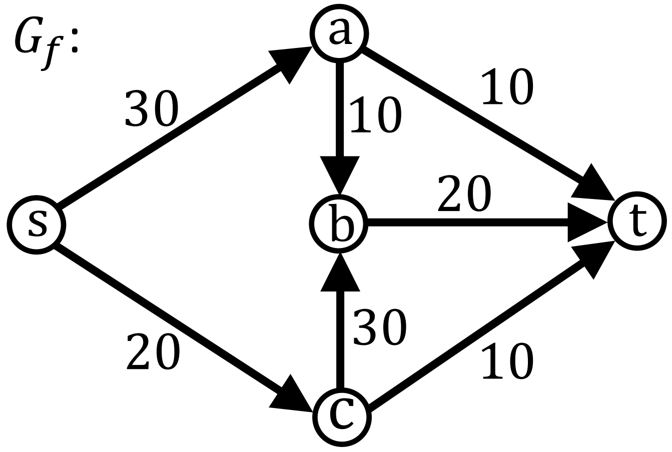
Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.
2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.
3. If an iteration starts with a valid flow, it ends with a valid flow.
   Each iteration sends flow along a residual $s - t$ path without violating capacities or conservation of flow. So, the resulting flow is valid.
4. The first iteration starts with a valid flow.

Max Flow Algorithm







Algorithm Overview

1. Start with 0 flow and initial residual graph.
2. Select an $s-t$ path P in residual graph.
3. Push bottleneck(P) flow on P.
4. Update residual graph.
5. Repeat until no $s-t$ paths exist in residual graph.

Ford-Fulkerson

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

   Each iteration sends flow along a residual $s - t$ path without violating capacities or conservation of flow. So, the resulting flow is valid.

4. The first iteration starts with a valid flow.

   0 flow on all edges meets capacity and conservation of flow constraints.

Ford-Fulkerson

What can be concluded from all this?

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.
   Each iteration sends flow along a residual $s - t$ path without violating capacities or conservation of flow. So, the resulting flow is valid.

4. The first iteration starts with a valid flow.
   0 flow on all edges meets capacity and conservation of flow constraints.

Ford-Fulkerson

What can be concluded from all this?

1. Ford-Fulkerson returns a valid flow.

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.
   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.
   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.
   Each iteration sends flow along a residual $s - t$ path without violating capacities or conservation of flow. So, the resulting flow is valid.

4. The first iteration starts with a valid flow.
   0 flow on all edges meets capacity and conservation of flow constraints.

Ford-Fulkerson

What can be concluded from all this?

1. Ford-Fulkerson returns a valid flow.

2. The running time is in $\Omega(|\text{Max Flow}|)$.

Claims:

1. If edge capacities are integer-valued, flows found by algorithm will be too.

   Bottleneck values will always be integers, so resulting flows are too.

2. If edge capacities are integer-valued, the algorithm will terminate.

   With integer capacities, each iteration increases flow by ≥ 1. Since the max flow is bounded (e.g., by total capacity into t), the algorithm needs $\leq$ |Max Flow| iterations.

3. If an iteration starts with a valid flow, it ends with a valid flow.

   Each iteration sends flow along a residual $s - t$ path without violating capacities or conservation of flow. So, the resulting flow is valid.

4. The first iteration starts with a valid flow.

   0 flow on all edges meets capacity and conservation of flow constraints.

Ford-Fulkerson – Running Time

Assuming integer edge capacities:

While loop runs at most ??? times.

```
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 – Running Time

Assuming integer edge capacities:

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

```
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
```

Ford-Fulkerson – 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
```

Ford-Fulkerson – 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
```

Ford-Fulkerson – 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
```

Ford-Fulkerson – 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: $O(|V|)$

```
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
```

Ford-Fulkerson – 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: $O(|V|)$
- 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
```

Ford-Fulkerson – 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: $O(|V|)$
- Update G_f (for each $e \in$ P, 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
```

Ford-Fulkerson – 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: $O(|V|)$

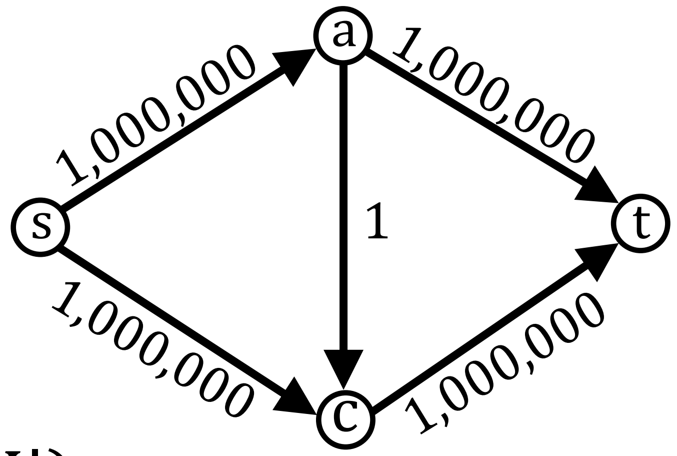
Update G_f (for each $e \in$ P, make e and e' $\in G_f$): $O(|E|)$

Total = $O(|f_{OPT}|\ 2\ (|E| + |V|))$ = $O(|E| \cdot |f_{OPT}|)$

```
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 – 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: $O(|V|)$

Update G_f (for each $e \in$ P, make e and $e' \in G_f$): $O(|E|)$

Total = $O(|f_{OPT}|\ 2\ (|E| + |V|)) = O(|E| \cdot |f_{OPT}|)$

```
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

```
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
```

Edmonds-Karp

```
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 = 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 – Running Time

```
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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in G_f exists
      P = shortest 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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Intuition: Paths change in the residual graph as we remove edges by filling them up or adding (back) edges by using forward edges for the first time.

```
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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Intuition: Adding flow to a residual graph can only make paths longer by removing (saturated) edges or adding back edges, which to be able to use, first requires reaching its head (at least as far as before) plus one extra hop.

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in G_f exists
      P = shortest 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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

```
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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

Intuition: An edge cannot be a bottleneck again without the shortest distance increasing, which can happen at most $|V| - 1$ times.

```
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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

Thus, each iteration needs a bottleneck + at most $|E| * O(|V|)$ bottlenecks

```
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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

Thus, each iteration needs a bottleneck + at most $|E| * O(|V|)$ bottlenecks

$\Rightarrow$ at most $O(|E||V|)$ iterations $\Rightarrow$

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in G_f exists
      P = shortest 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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

Thus, each iteration needs a bottleneck + at most $|E| * O(|V|)$ bottlenecks

$\Rightarrow$ at most $O(|E||V|)$ iterations $\Rightarrow O(|E|^2|V|)$ time total.

```
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
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

Th

Running Time:

$O(|E| \cdot |f_{OPT}|)$ [Ford-Fulkerson, 1956]

$O(|V||E|^2)$ [Edmonds-Karp, 1972]

Ma

ge

```
   f'= augment(f, P)
   f = f'
   G_f = G_f'
return f
```

```
      f((v, u)) -= b
    else
      f((u, v)) += b
  return f
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

Th

Ma

Running Time:

$O(|E| \cdot |f_{OPT}|)$ [Ford-Fulkerson, 1956]

$O(|V||E|^2)$ [Edmonds-Karp, 1972]

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

ge

```
   f'= augment(f, P)
   f = f'
   Gf = Gf'
return f
```

```
      f((v, u)) -= b
    else
      f((u, v)) += b
  return f
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

Th

Running Time:

$O(|E| \cdot |f_{OPT}|)$ [Ford-Fulkerson, 1956]

$O(|V||E|^2)$ [Edmonds-Karp, 1972]

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

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

Ma

ge

```
   f'= augment(f, P)
   f = f'
   Gf = Gf'
return f
```

```
      f((v, u)) -= b
    else
      f((u, v)) += b
return f
```

Edmonds-Karp – Running Time

Claim 1: The distance of the shortest path from s to any v in the residual graph is non-decreasing over the iterations of the algorithm.

Claim 2: Each edge can be the bottleneck for a path at most $O(|V|)$ times.

Th

Running Time:

$$\left.\begin{array}{l} O(|E| \cdot |f_{OPT}|) \text{ [Ford-Fulkerson, 1956]} \\ O(|V||E|^2) \text{ [Edmonds-Karp, 1972]} \\ O(|V||E|) \text{ [Orlin, 2013]} \\ O\left(|E|^{1+o(1)}\right) \text{ [Chen et al., 2022]} \end{array}\right\} \in \boldsymbol{O}(\textbf{Max Flow})$$

Ma

ge

```
    f'= augment(f, P)
    f = f'
    Gf = Gf'
return f
```

```
      f((v, u)) -= b
    else
      f((u, v)) += b
return f
```