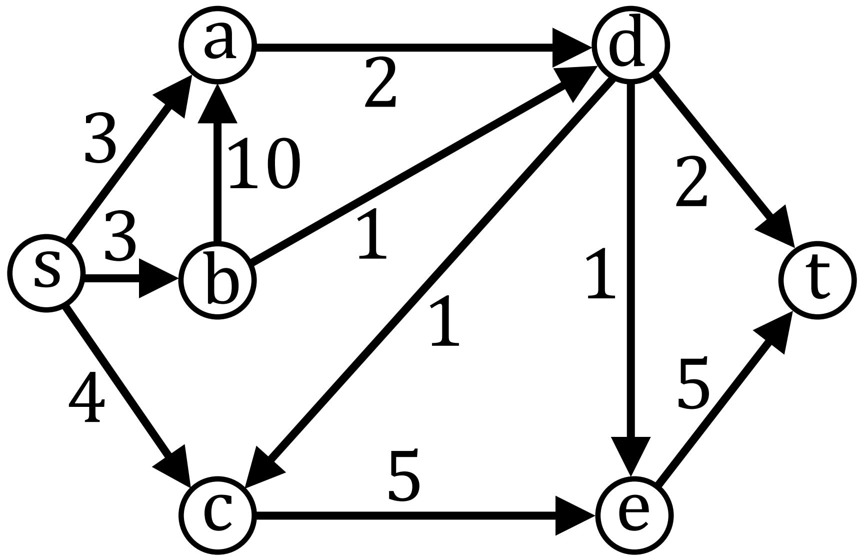
Flow Networks
CSCI 532

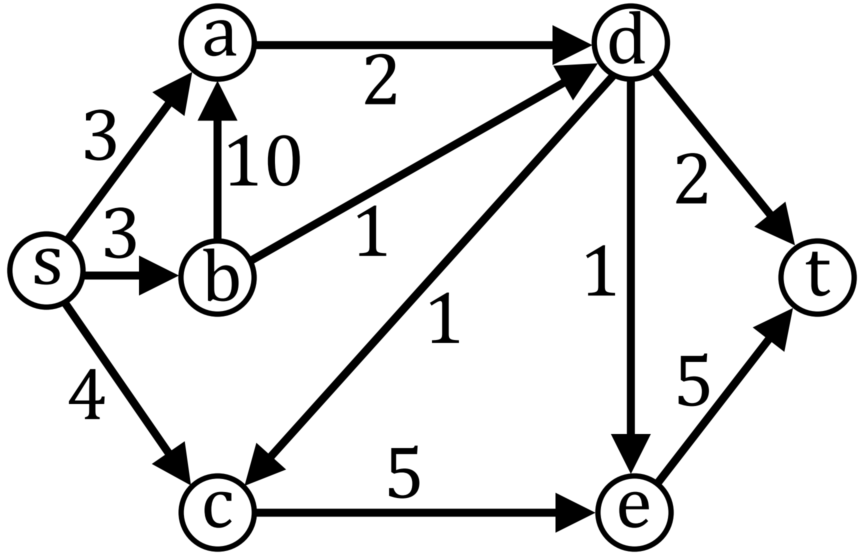
Motivation

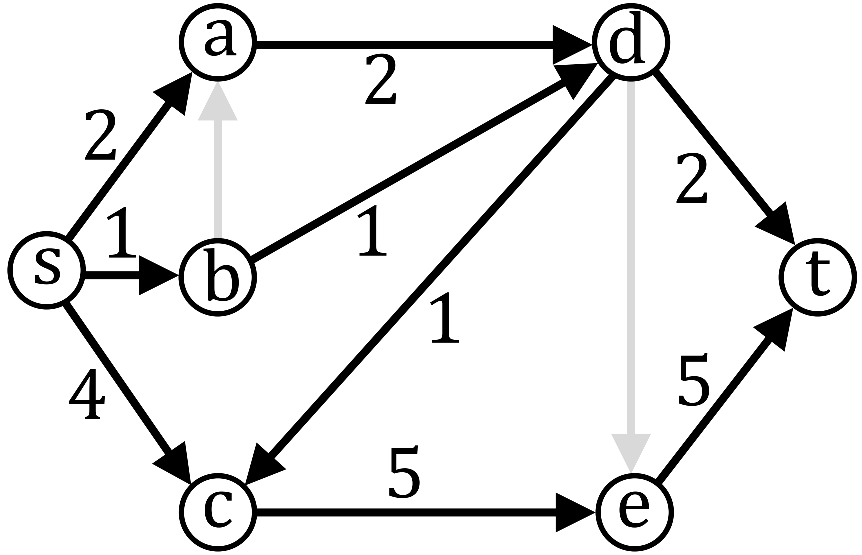
Suppose we have a directed graph that represent an oil pipeline network. Edge weight represent pipe capacity. How much oil can we transfer from source s to sink t?



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?





Total flow = 7

Flow Network

Flow Network:

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

Flow Network

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.

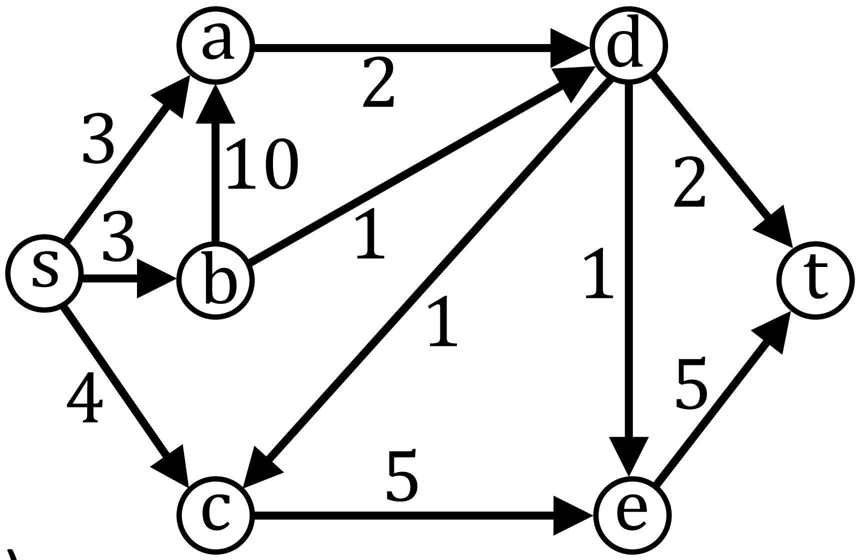
We'll also sometimes use the assumption that the capacities are positive integer values.

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

Flow Network

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.



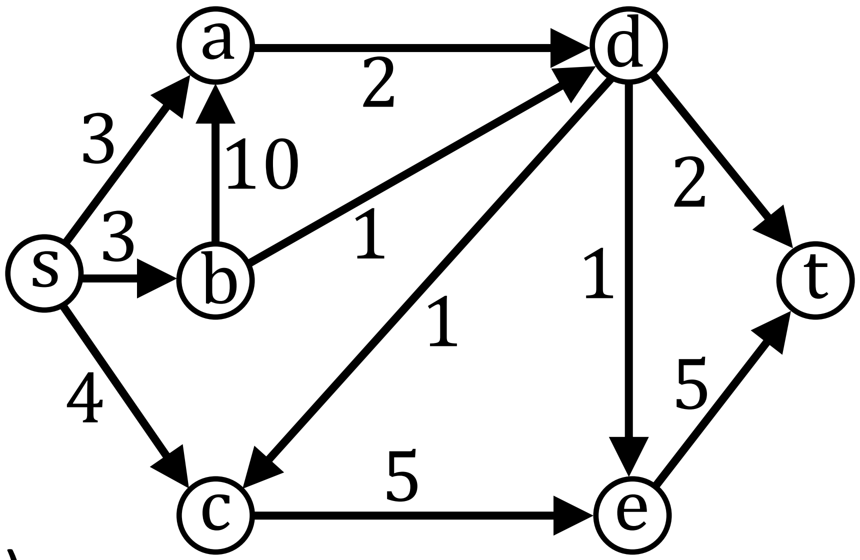
An $s - t$ flow is a function $f\colon E \to \mathbb{R}^+$ such that:

- $0 \leq f(e) \leq c_e, \forall e \in E$. (capacity constraint)

Flow Network

Flow Network:

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



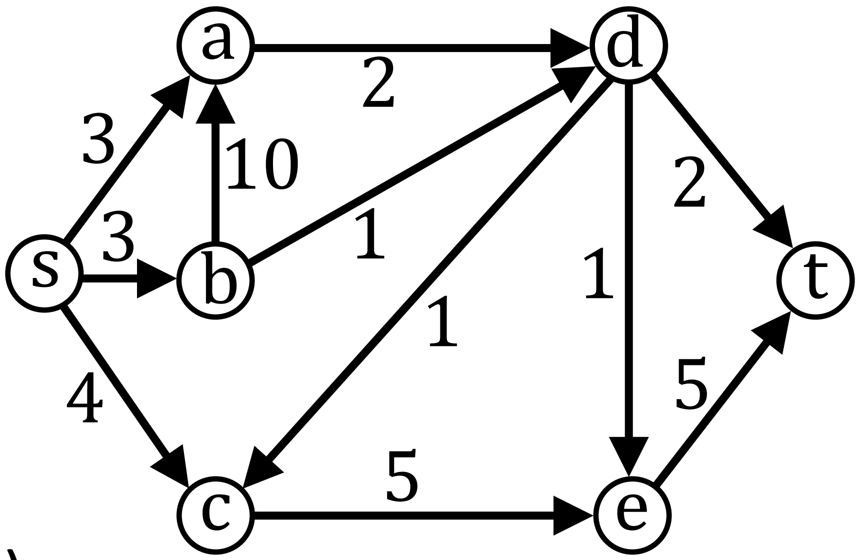
An $s - t$ flow is a function $f\colon E \rightarrow \mathbb{R}^+$ such that:

- $0 \leq f(e) \leq c_e, \forall e \in E$. (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: “Everything that goes into a node has to come out, except for s and t”)

Flow Network

Flow Network:

- Directed-edge graph, $G = (V, E)$.
- 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 $f\colon E \to \mathbb{R}^+$ such that:

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

Flow Network

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

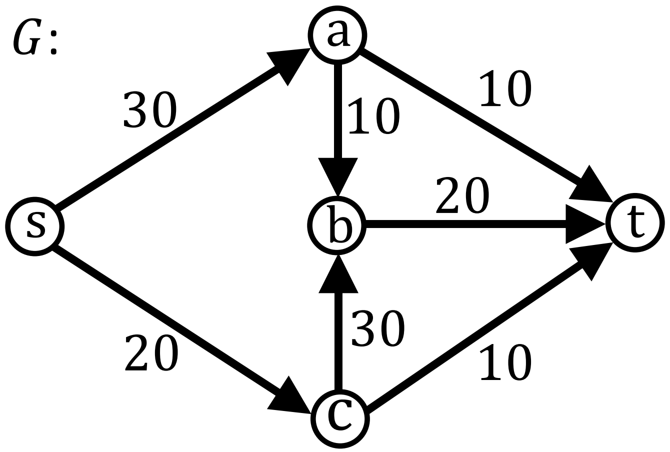
Flow Network:

- 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:

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

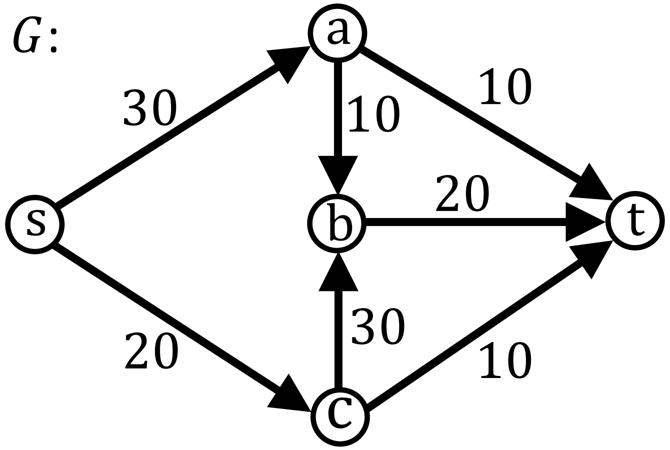
Max Flow Algorithm



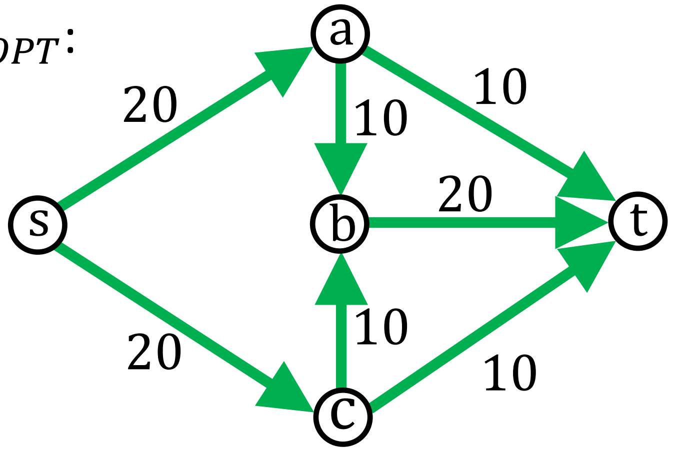
Max Flow?

Max Flow Algorithm

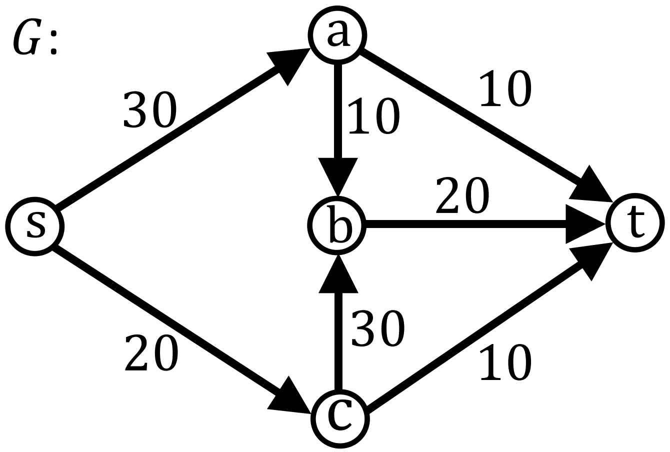
G:

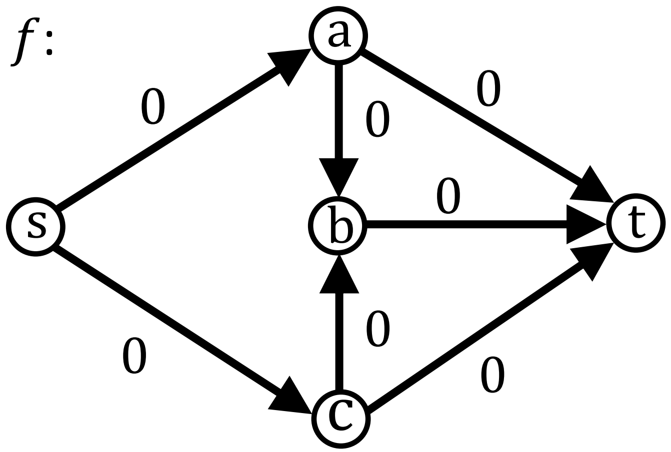


f_{OPT}:

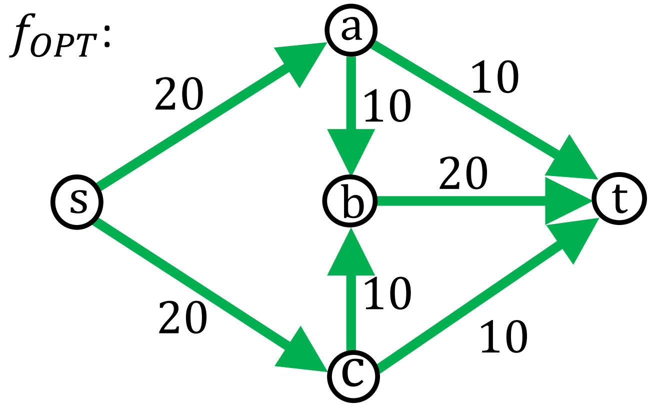


Max Flow Algorithm

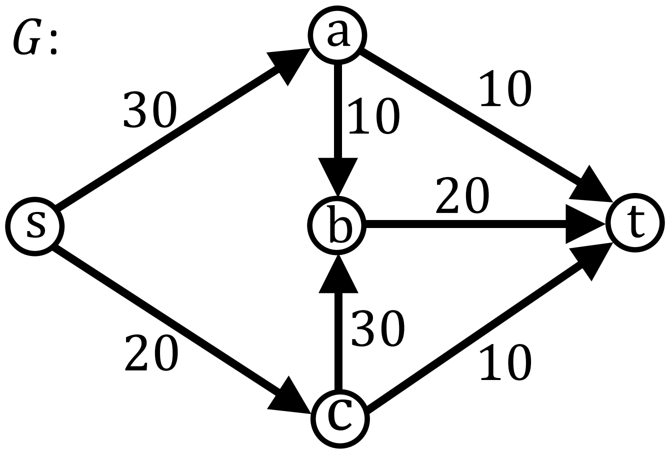


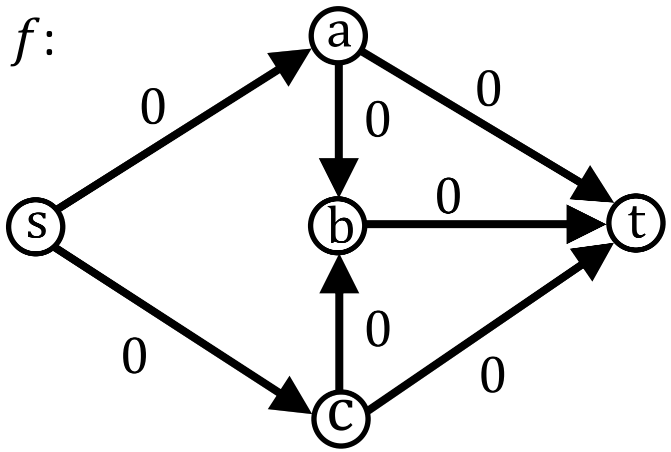


Ideas?



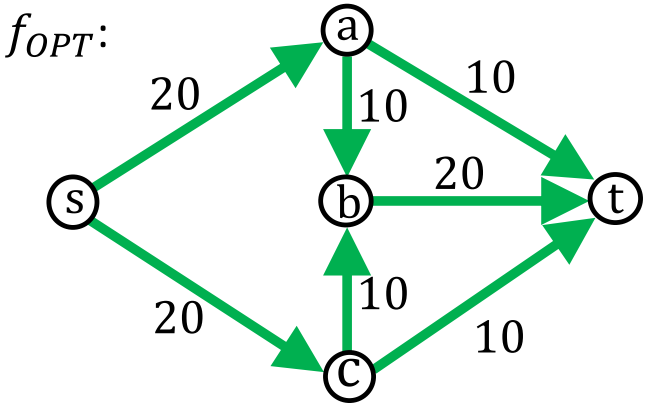
Max Flow Algorithm





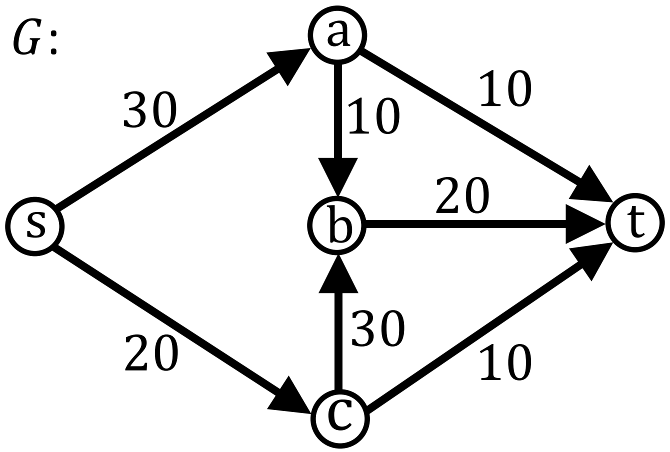
Ideas?

Somehow, we are going to have to put flow on edges. Should we select edges or paths?

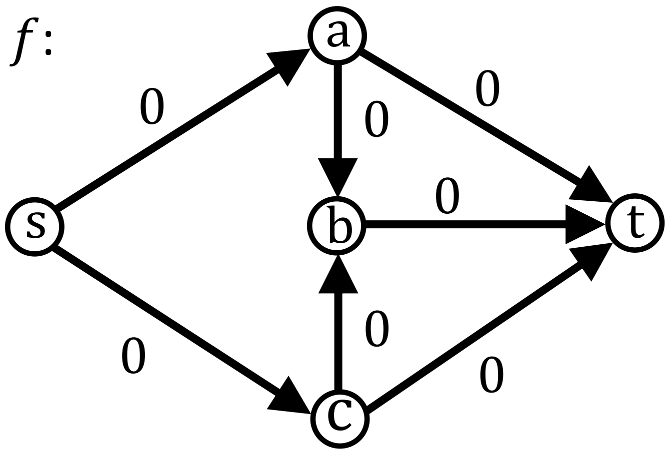


Max Flow Algorithm

G:



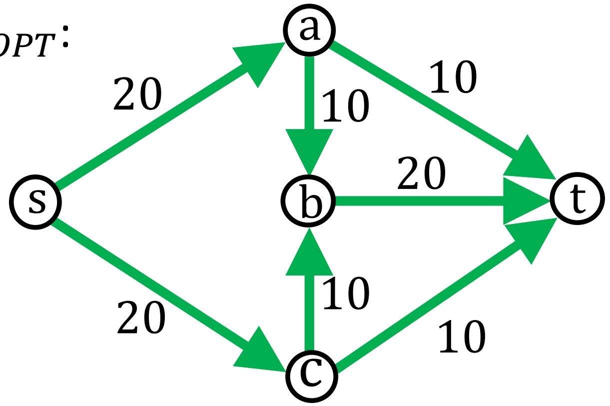
f:



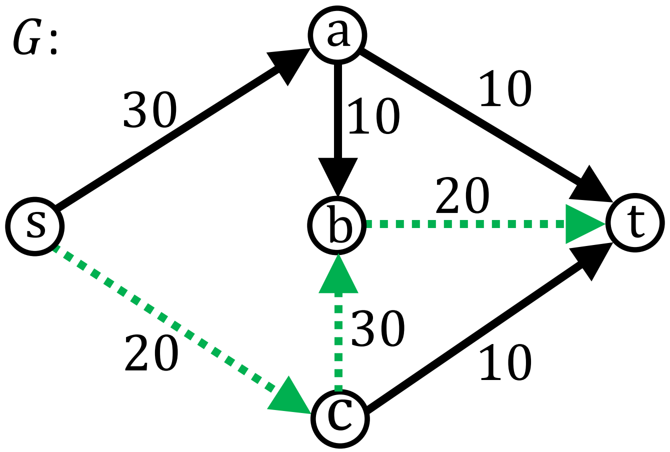
Ideas?

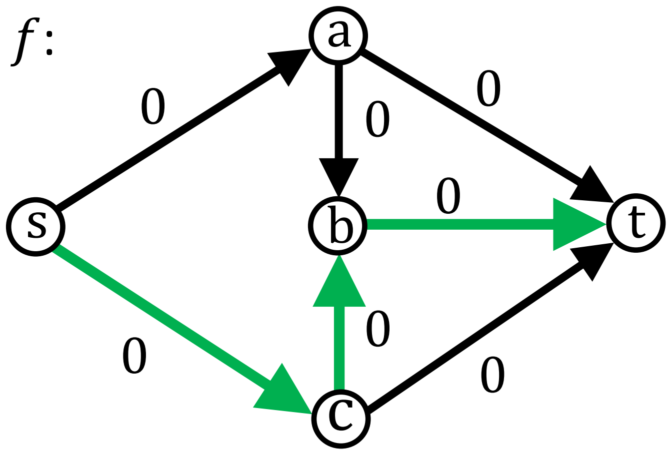
1. Select a path P.

f_{OPT}:



Max Flow Algorithm

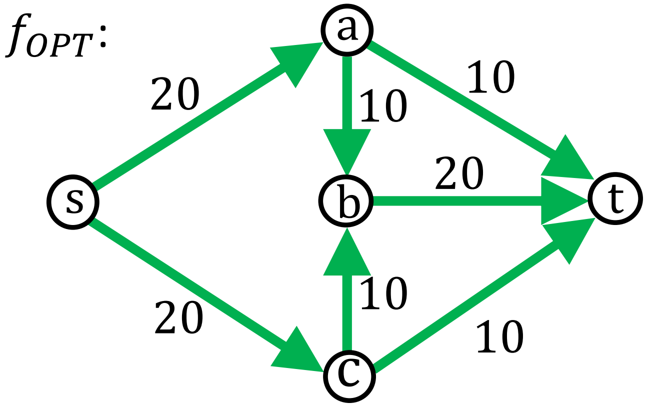




Ideas?

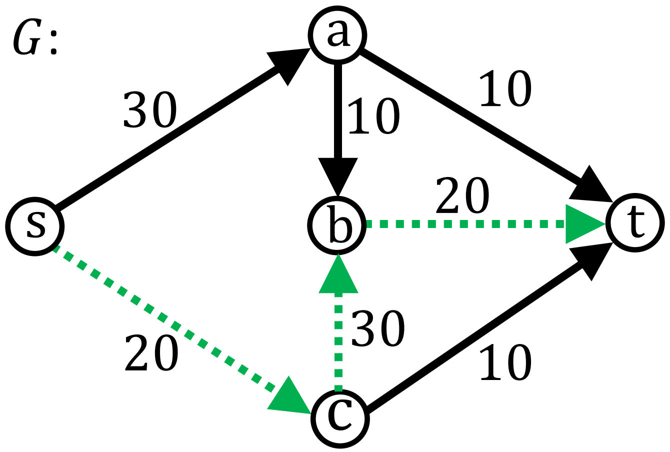
1. Select a path P.

How much flow should we push?

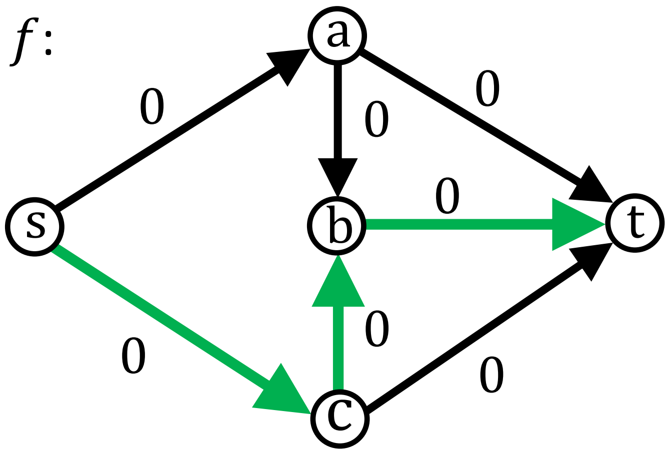


Max Flow Algorithm

G:



f:



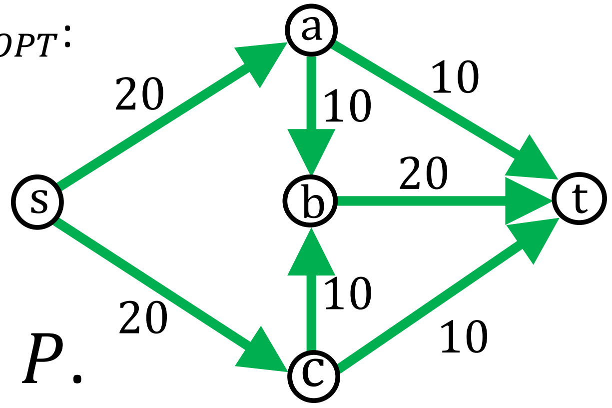
Ideas?

1. Select a path P.

How much flow should we push?

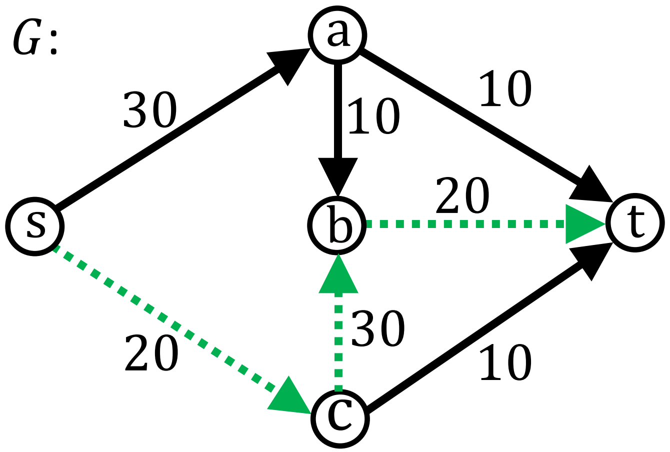
As much as possible.

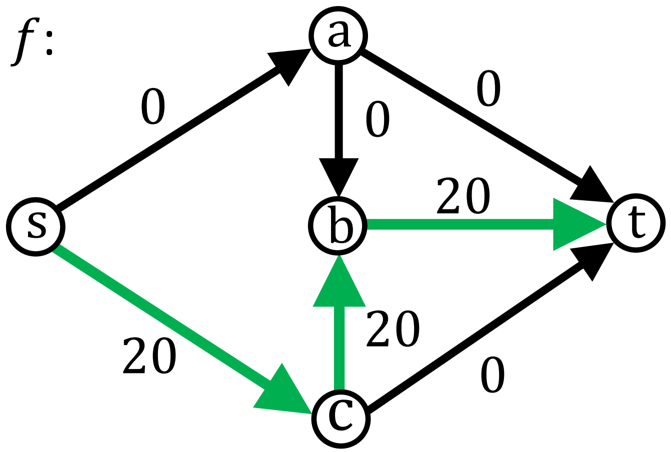
f_{OPT}:



`bottleneck(P)` = minimum capacity on any edge in path P.

Max Flow Algorithm





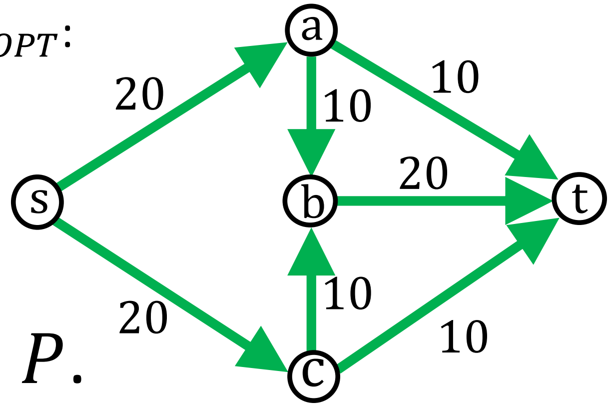
Ideas?

1. Select a path P.

How much flow should we push?

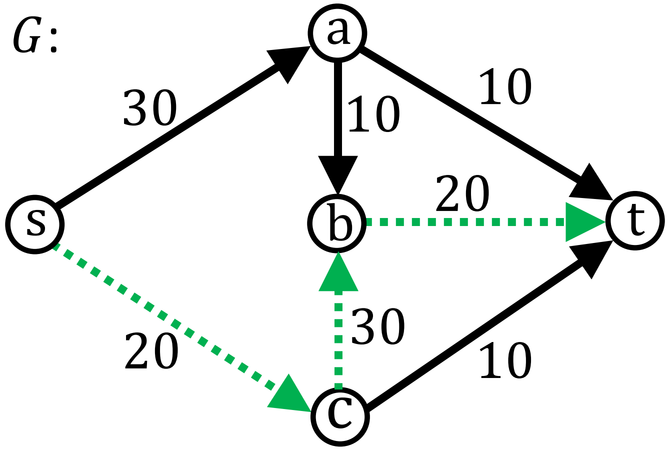
As much as possible.

`bottleneck(P)` = minimum capacity on any edge in path P.

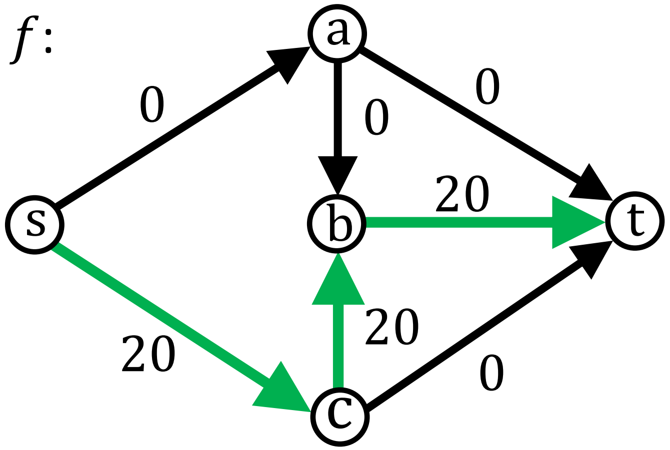


Max Flow Algorithm

G:



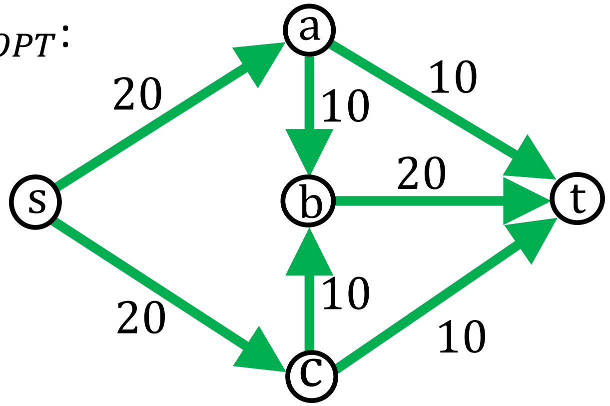
f:



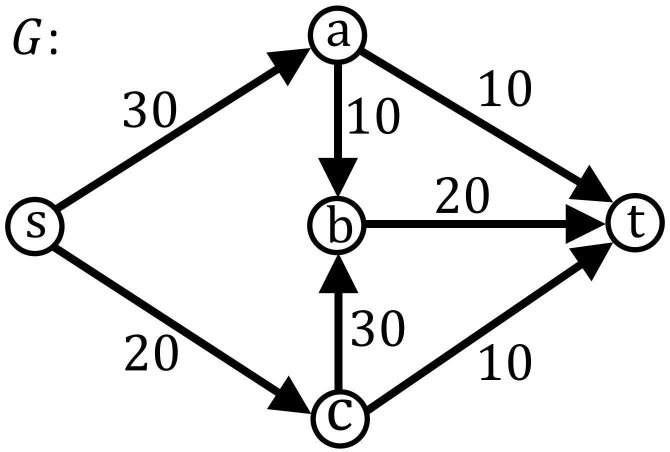
Ideas?

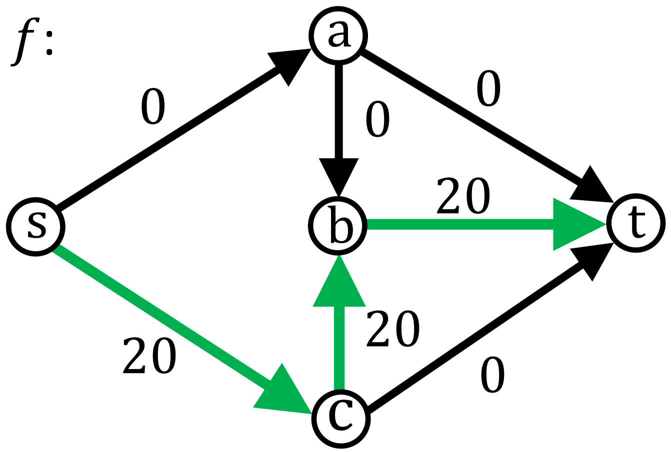
1. Select a path P.
2. Push bottleneck(P) flow on P.

f_{OPT}:



Max Flow Algorithm

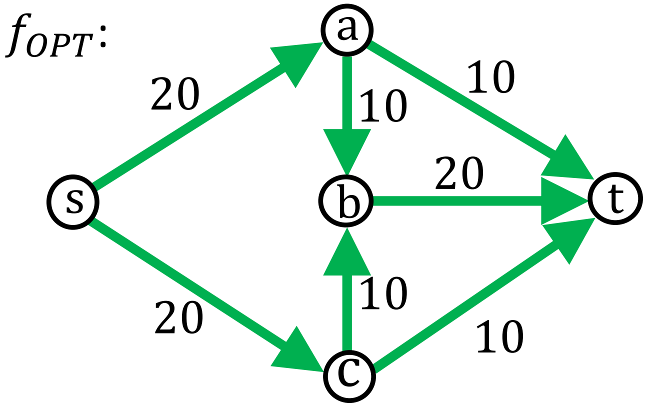




Ideas?

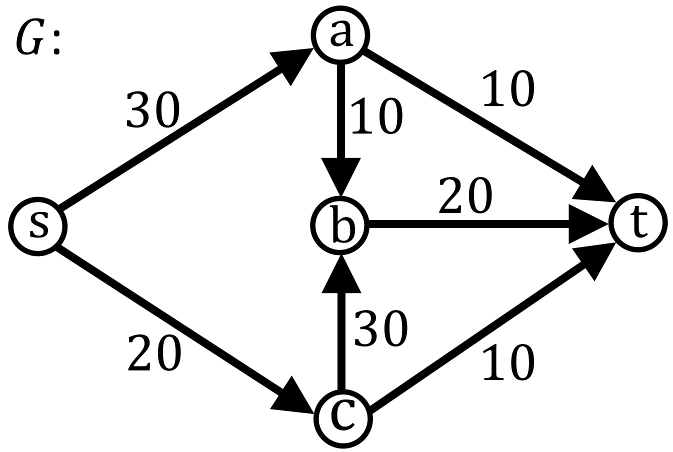
1. Select a path P.
2. Push bottleneck(P) flow on P.

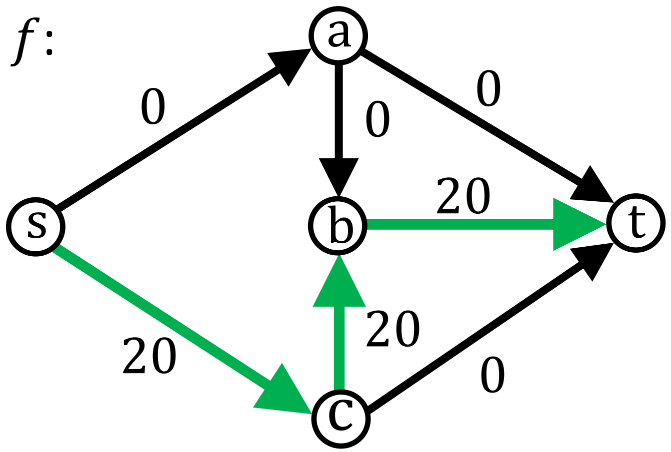
So far:
- **Guaranteed to meet conservation of flow constraints.**
- **Guaranteed to meet capacity constraints.**

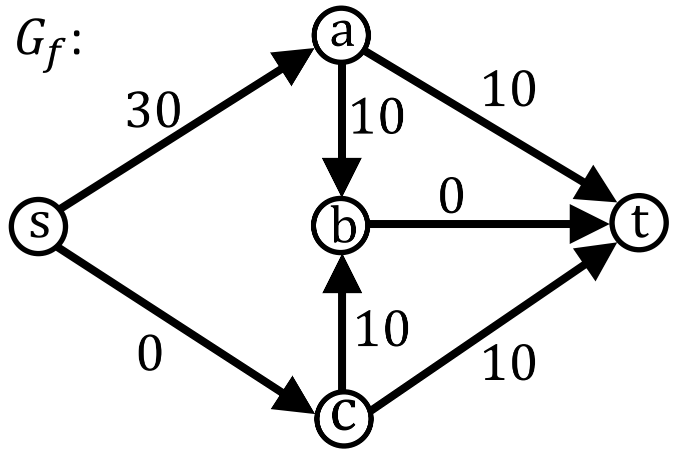


Max Flow Algorithm

Residual Capacity

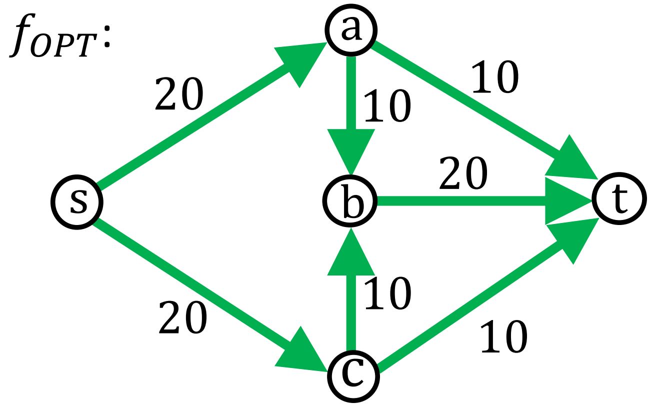




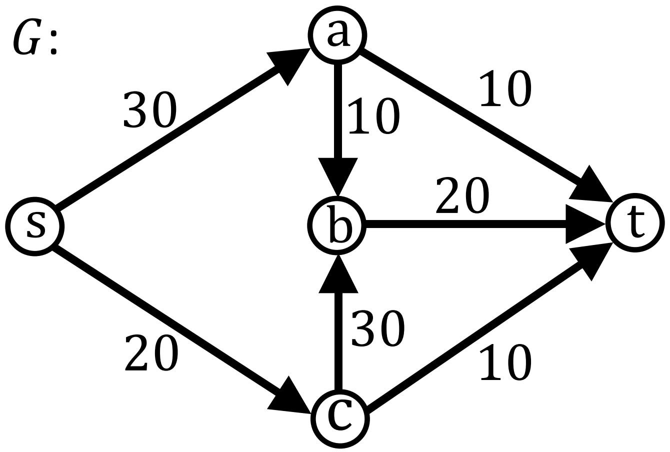


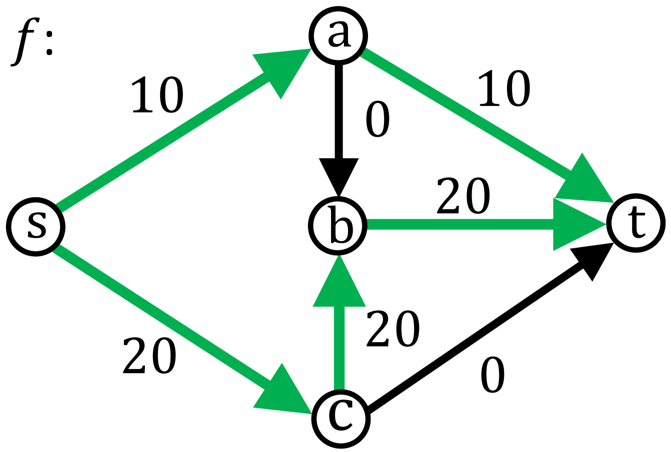
Ideas?

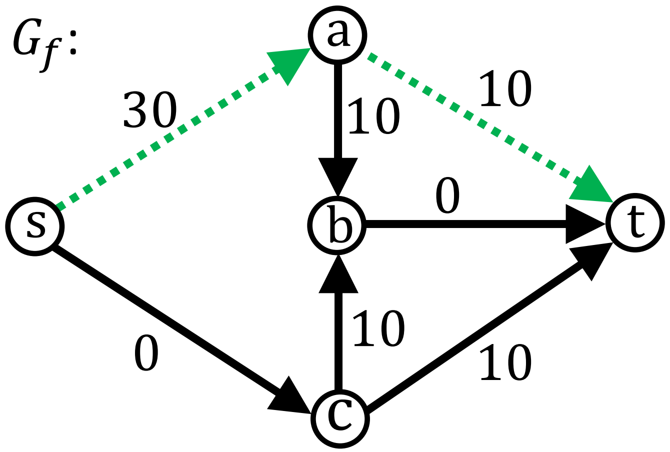
1. Select a path P.
2. Push bottleneck(P) flow on P.



Max Flow Algorithm

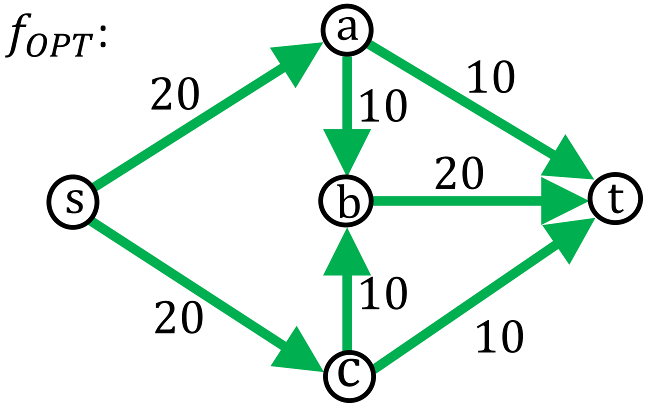




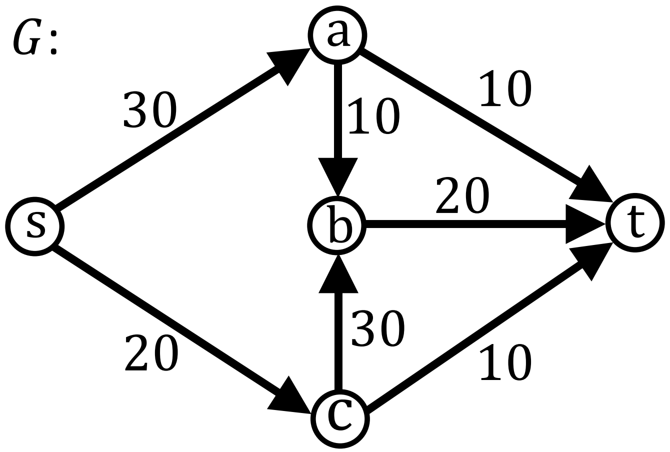


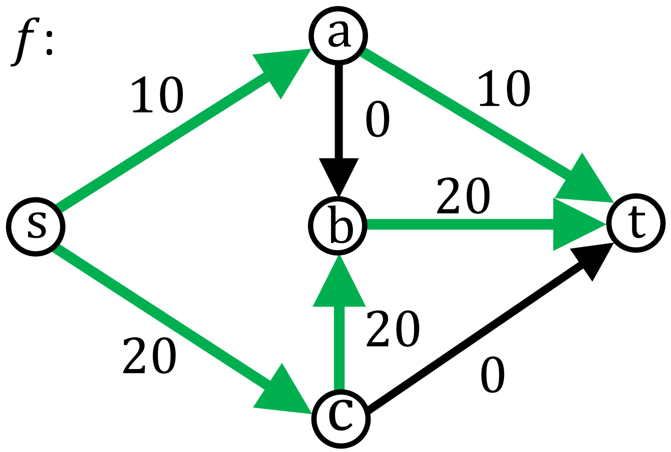
Ideas?

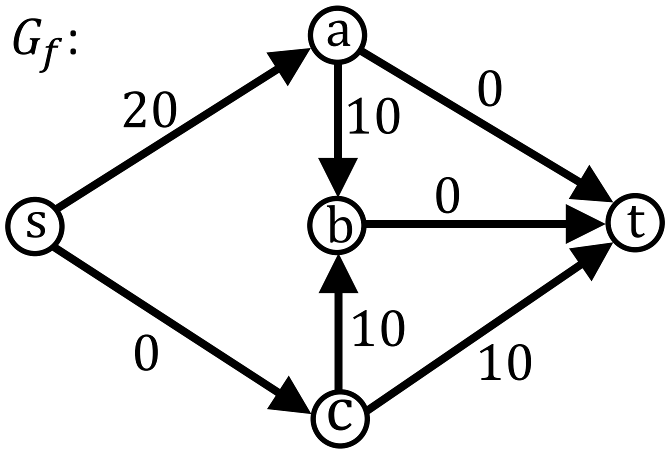
1. Select a path P.
2. Push bottleneck(P) flow on P.



Max Flow Algorithm

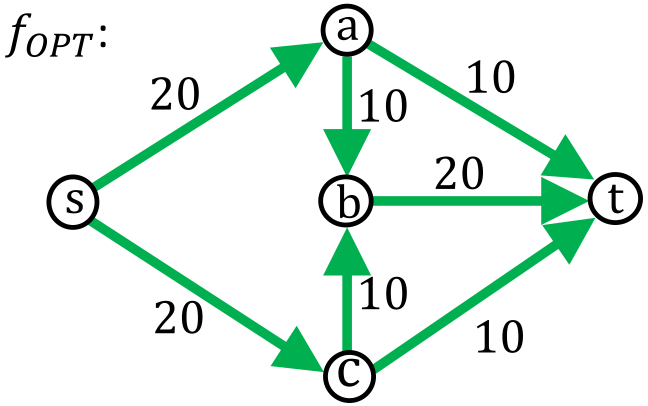




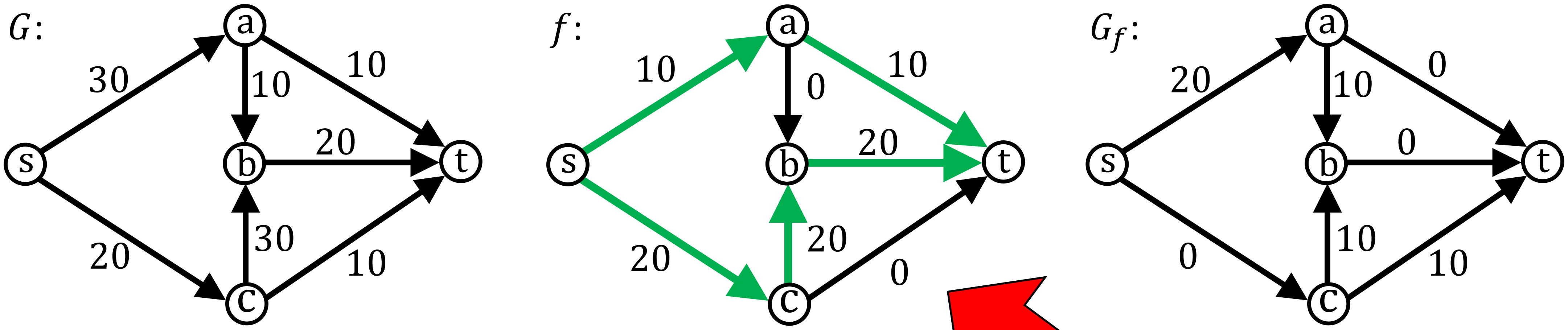


Ideas?

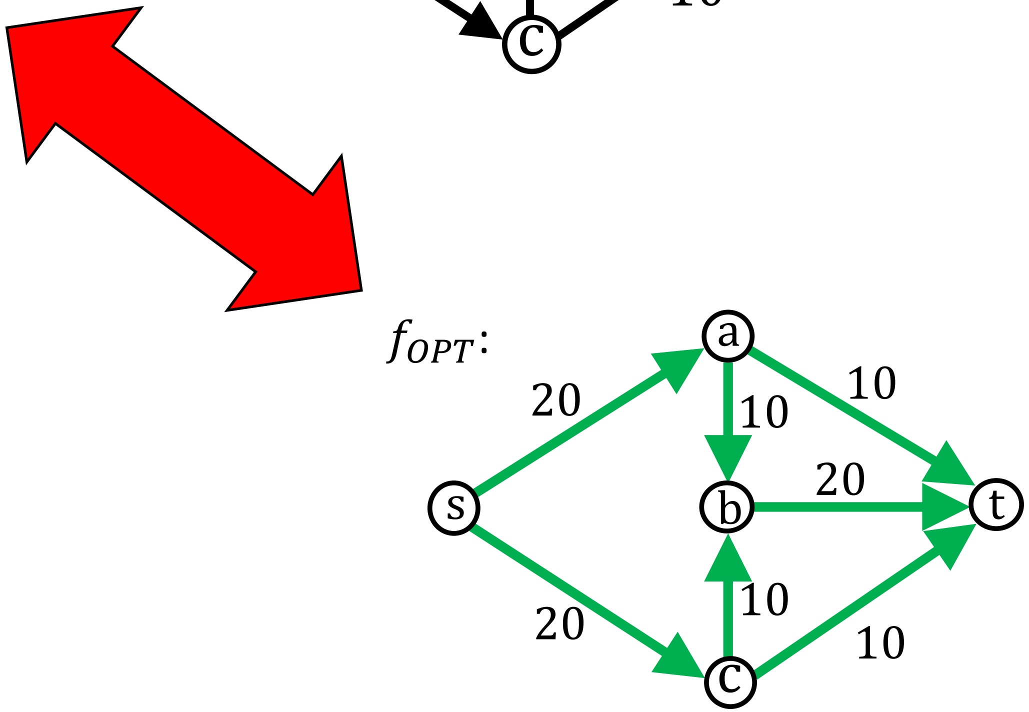
1. Select a path P.
2. Push bottleneck(P) flow on P.



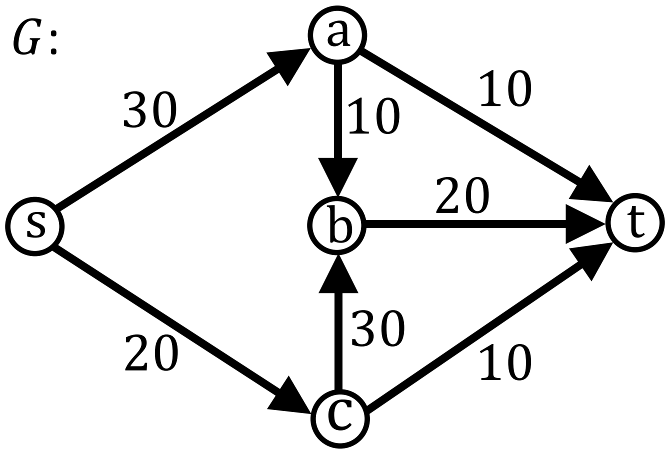
Max Flow Algorithm

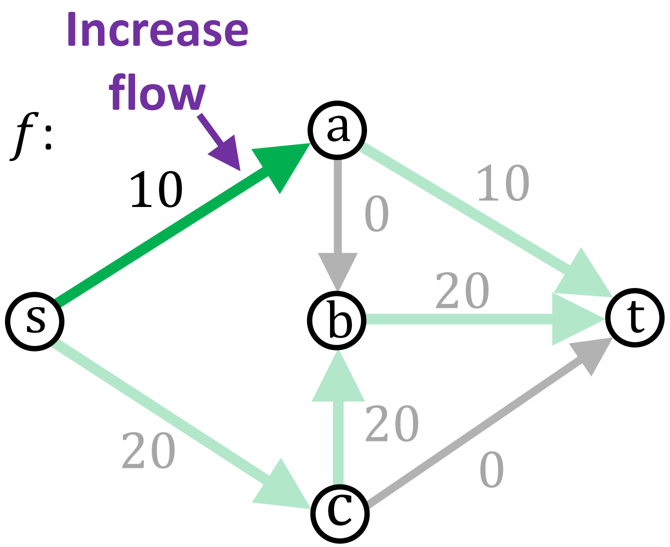


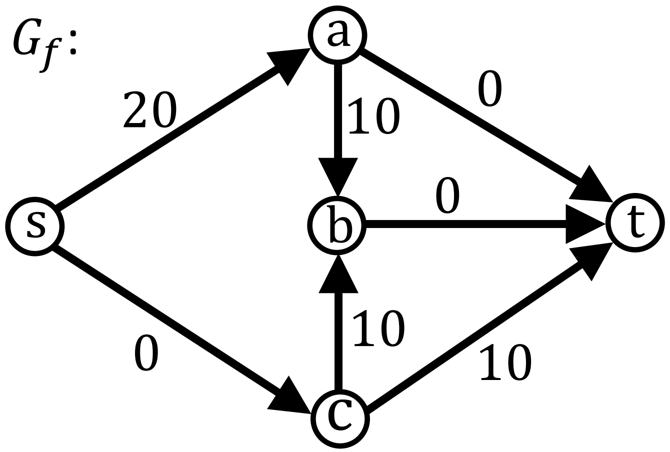
We need some way to reroute flow.



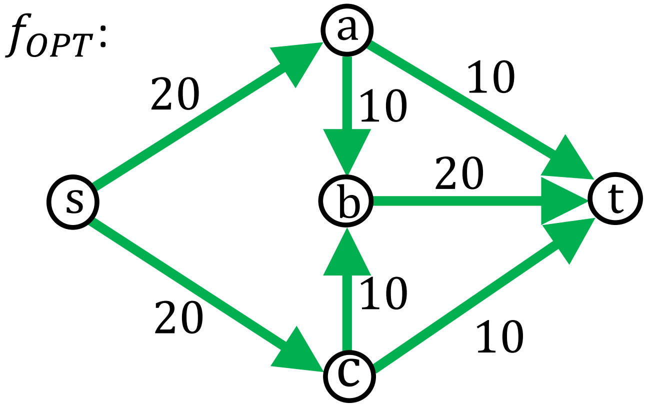
Max Flow Algorithm



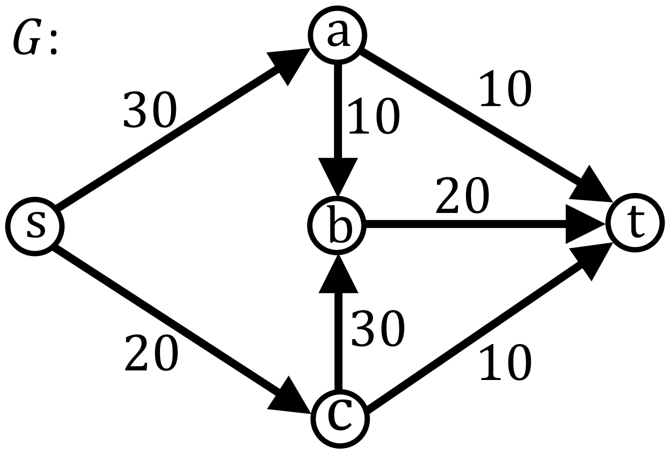


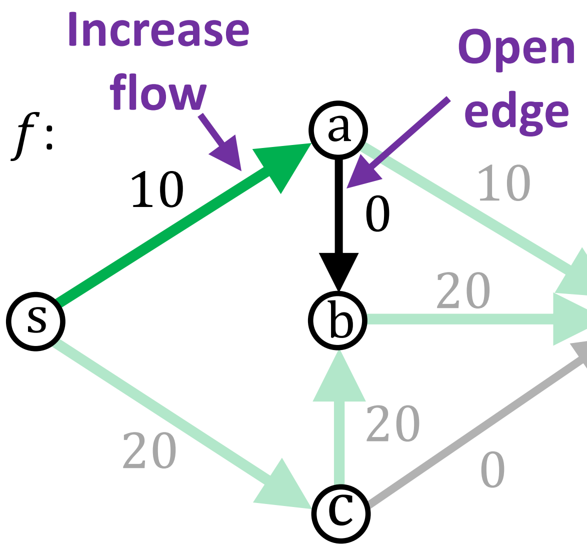


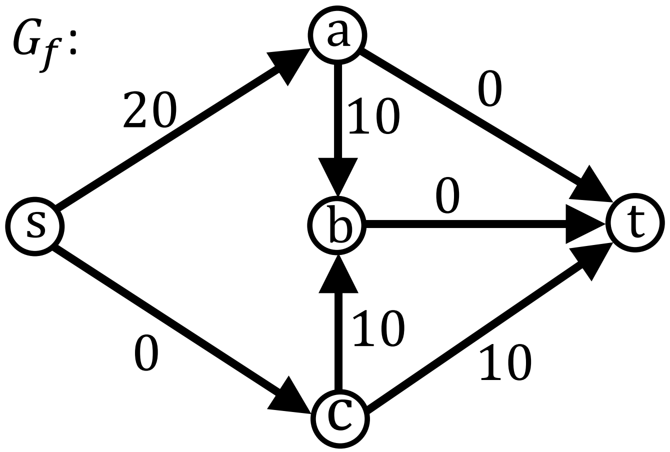
We need some way to reroute flow.



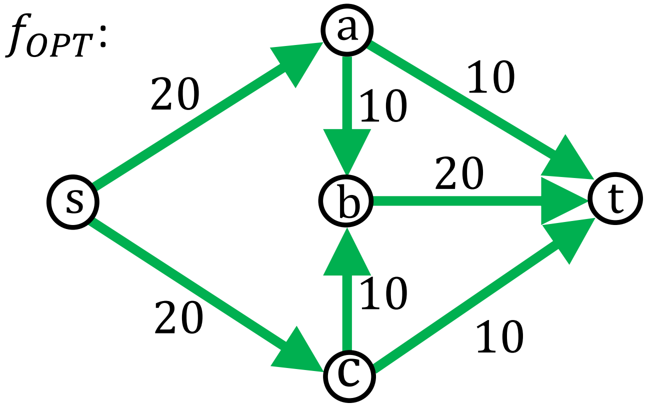
Max Flow Algorithm





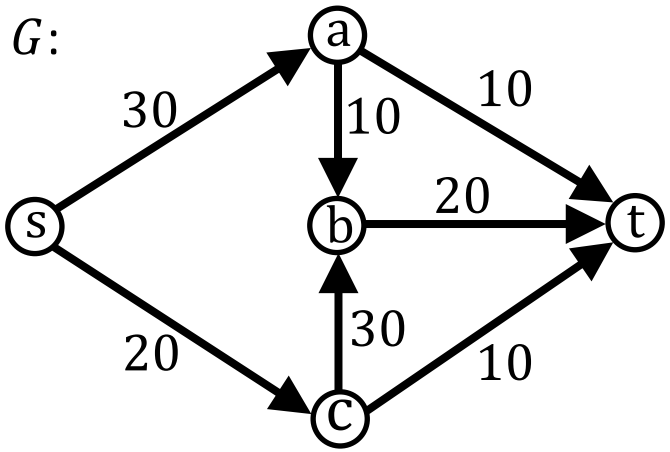


We need some way to reroute flow.

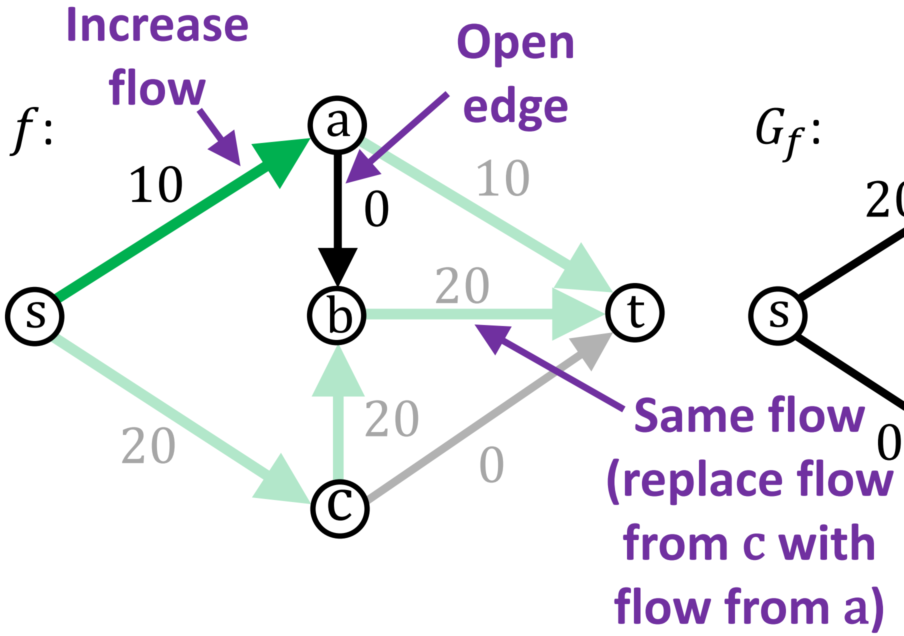


Max Flow Algorithm

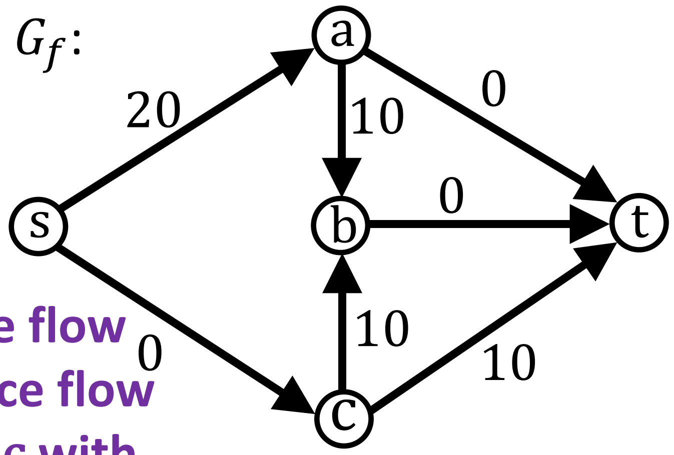
G:



f:

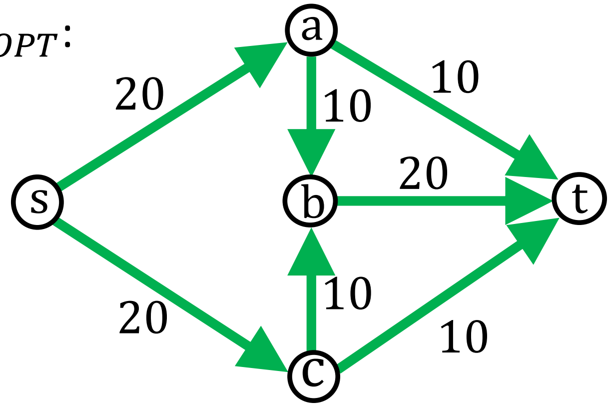


G_f:

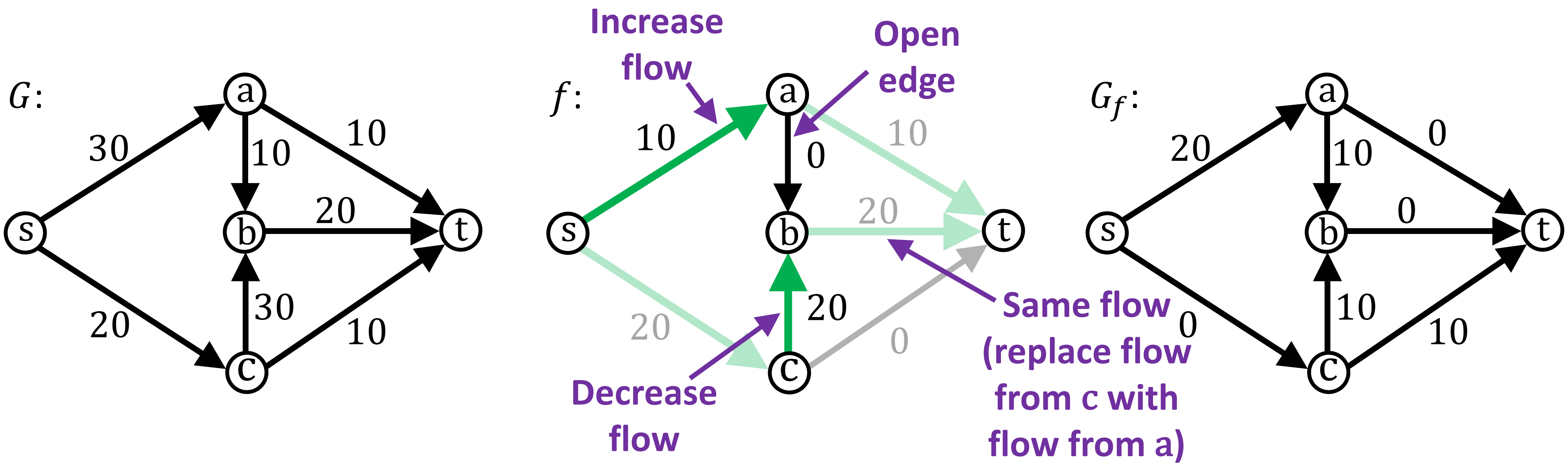


We need some way to reroute flow.

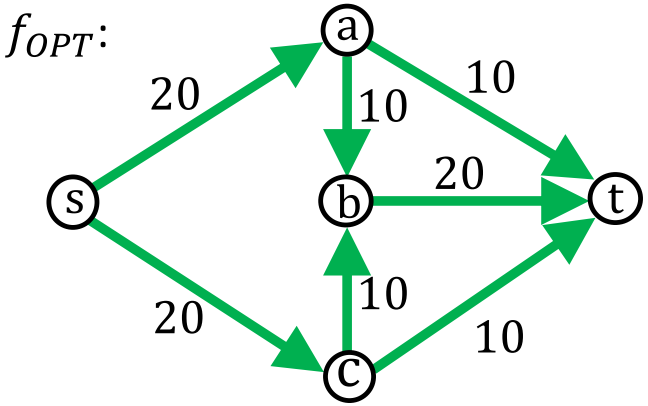
f_{OPT}:



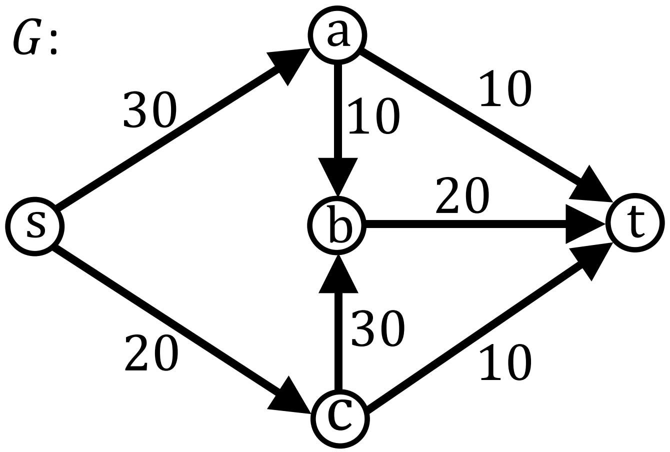
Max Flow Algorithm

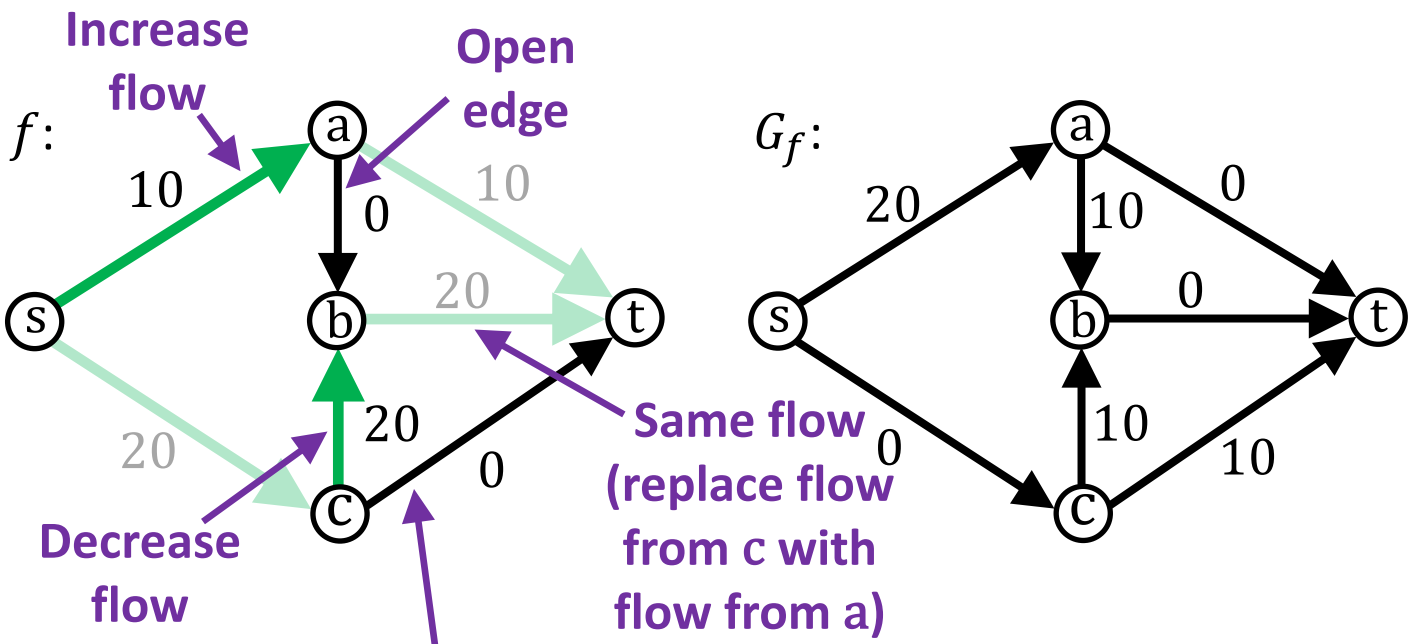


We need some way to reroute flow.



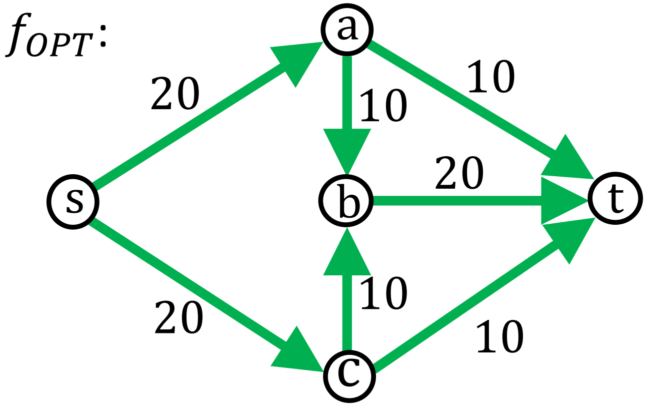
Max Flow Algorithm



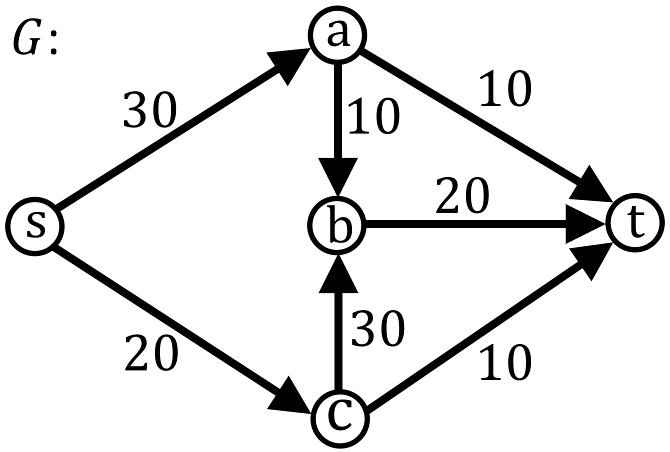


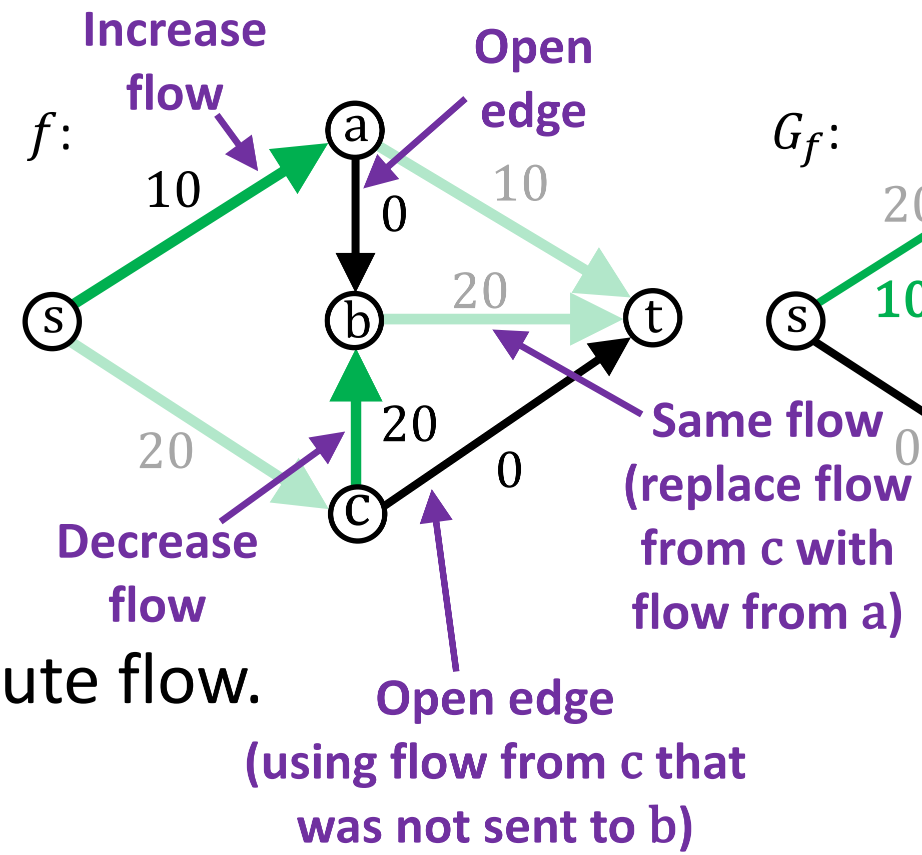
We need some way to reroute flow.

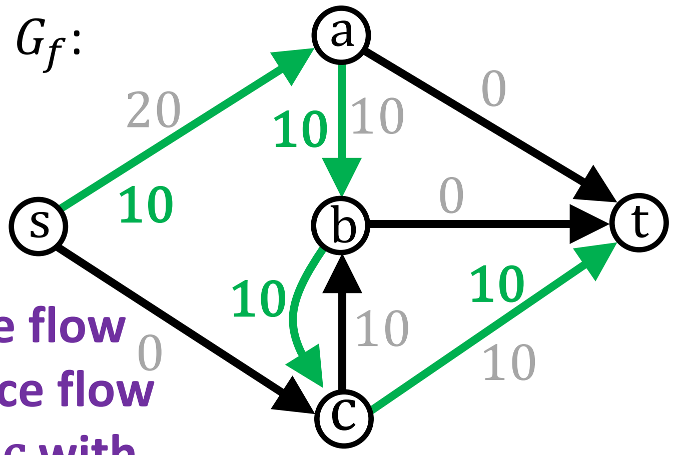
Open edge (using flow from c that was not sent to b)



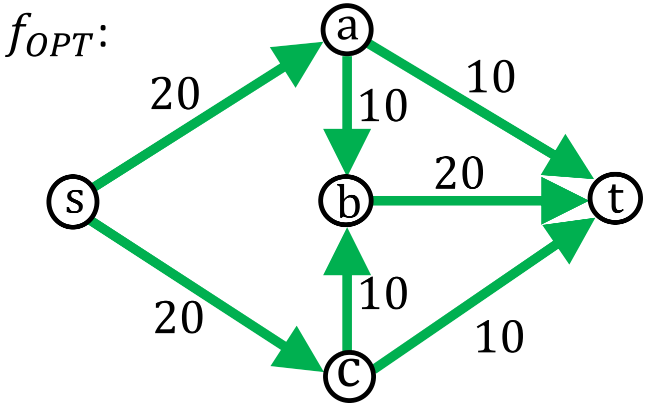
Max Flow Algorithm



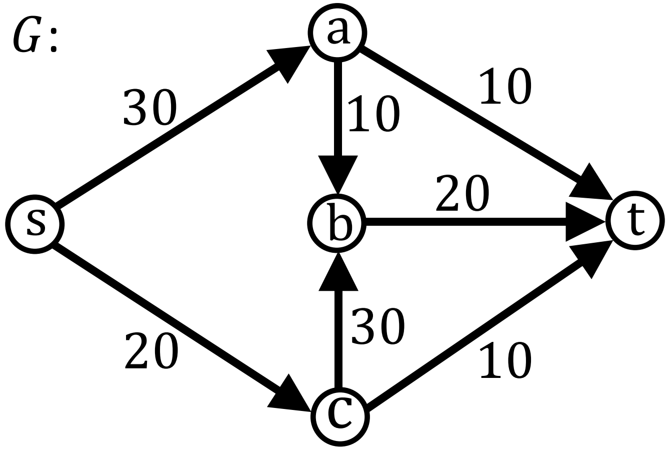


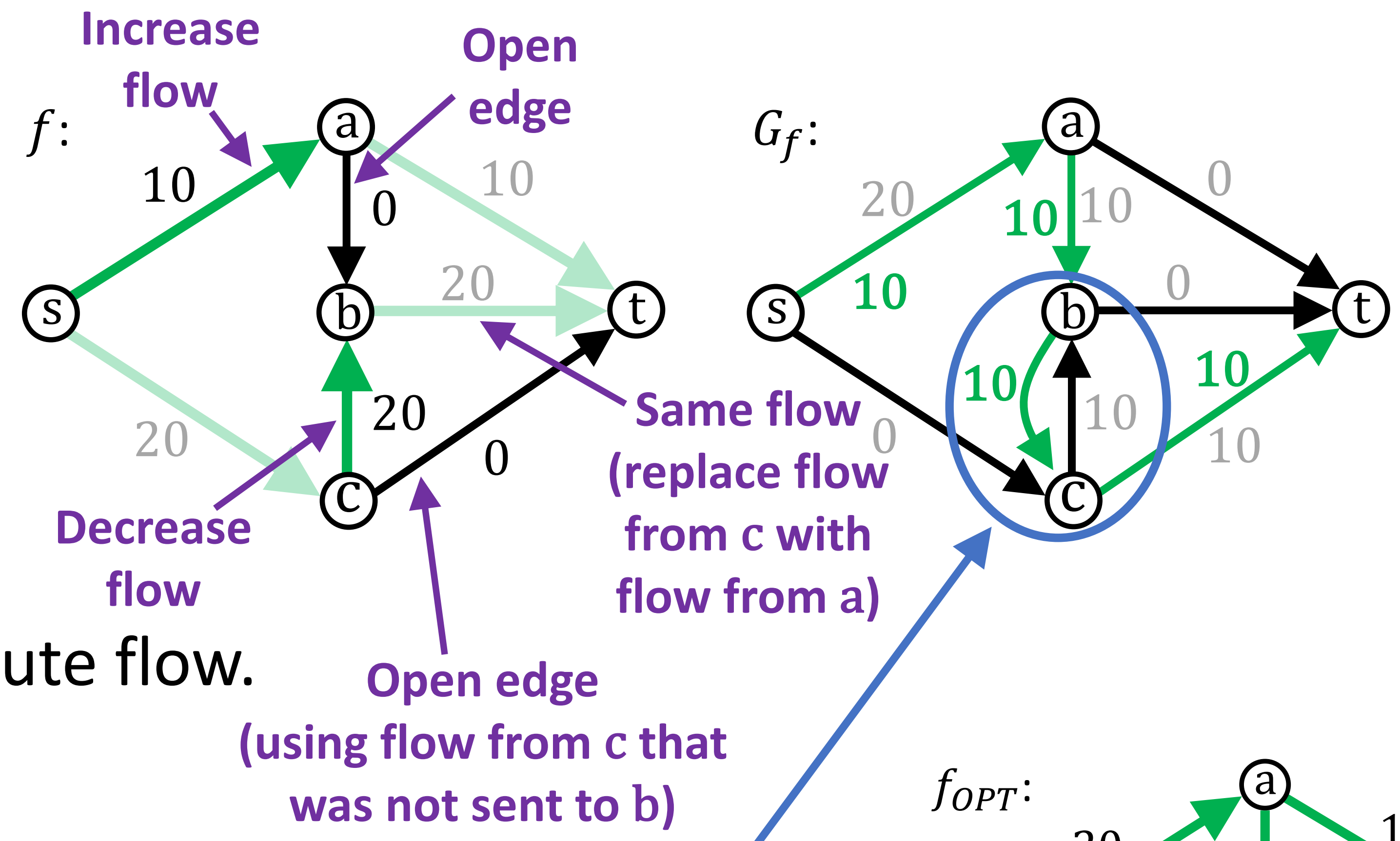


We need some way to reroute flow.

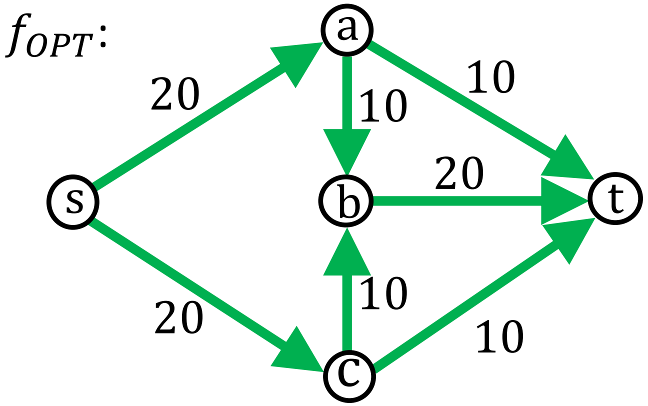


Max Flow Algorithm



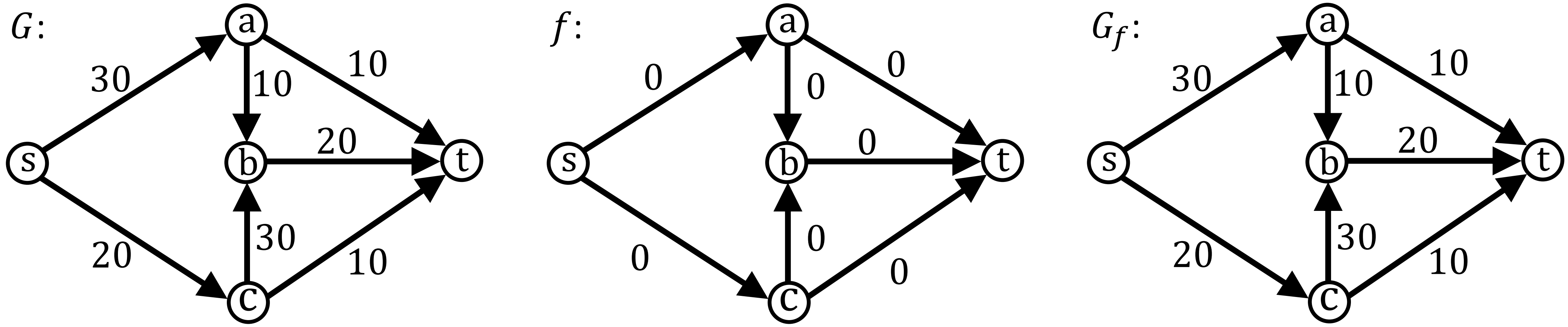


We need some way to reroute flow.



This edge does not physically exist but exists virtually as a mechanism to reroute flow that has already been pushed.

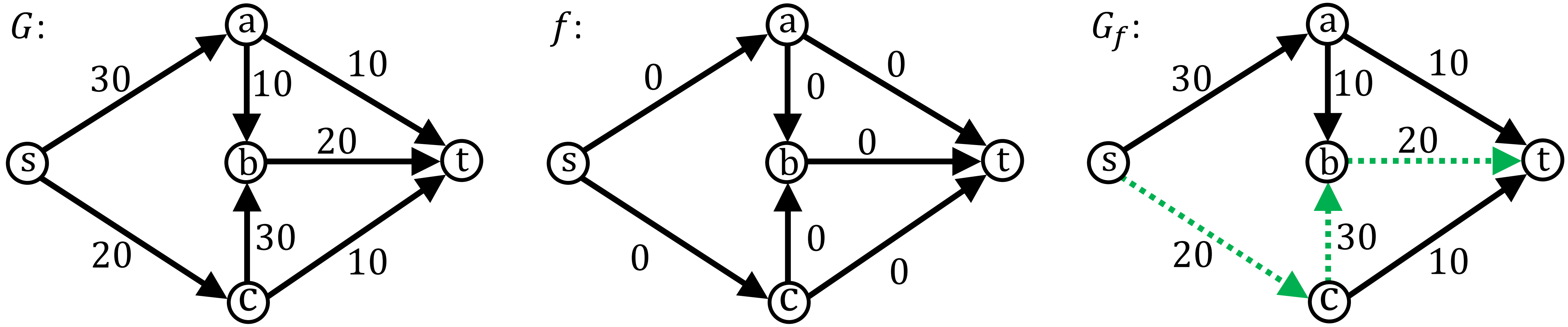
Max Flow Algorithm



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.

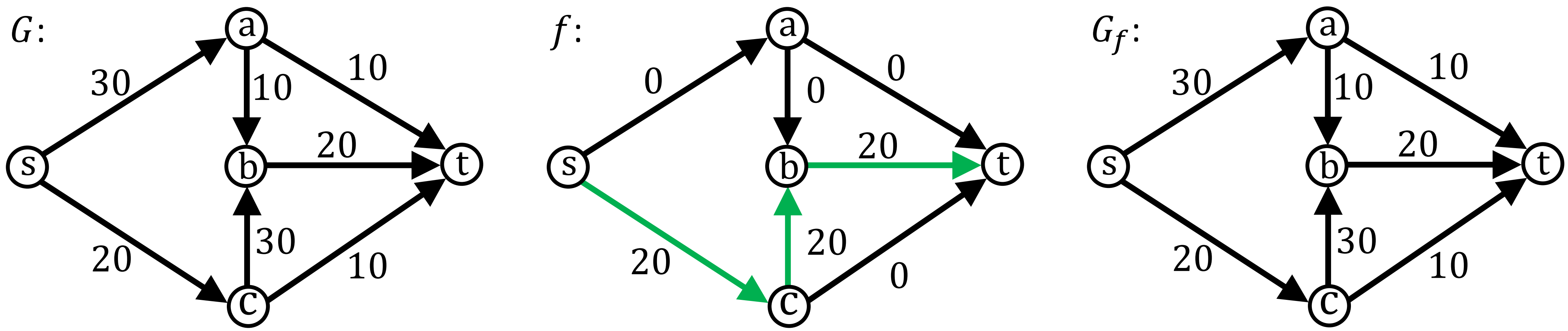
Max Flow Algorithm



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.

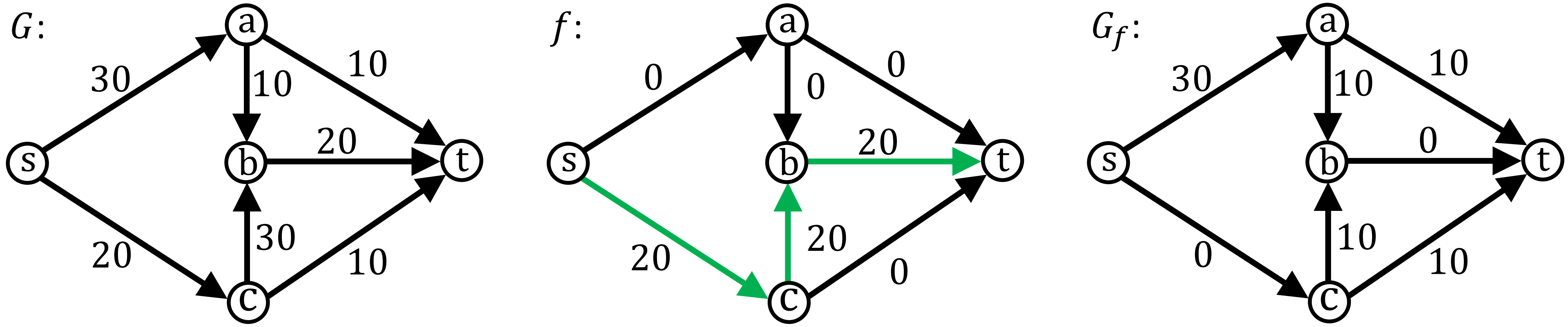
Max Flow Algorithm



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.

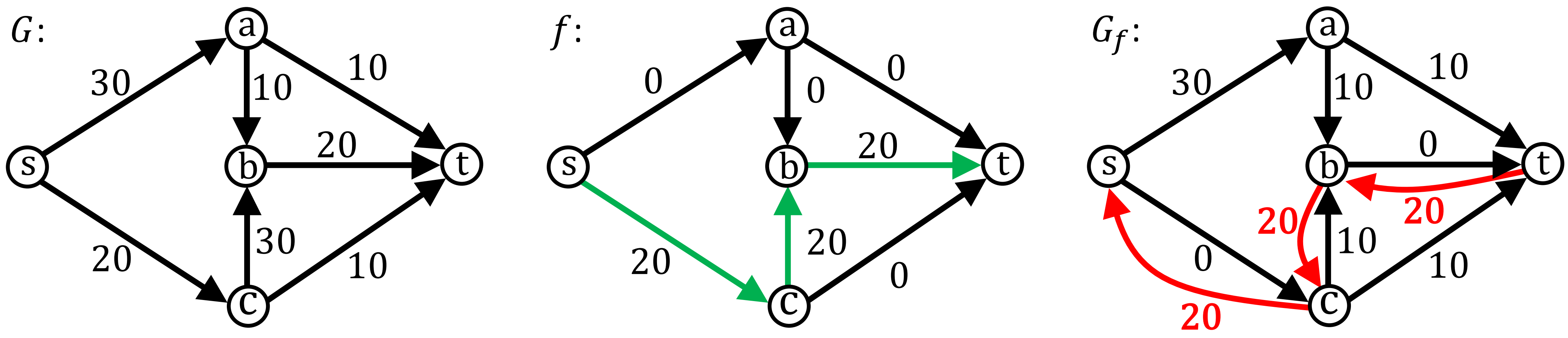
Max Flow Algorithm



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.

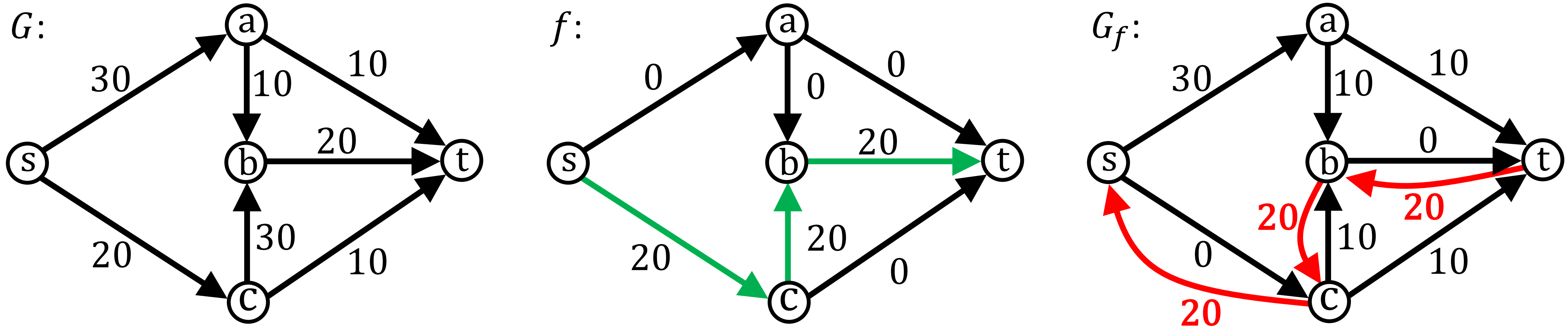
Max Flow Algorithm



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.

Max Flow Algorithm

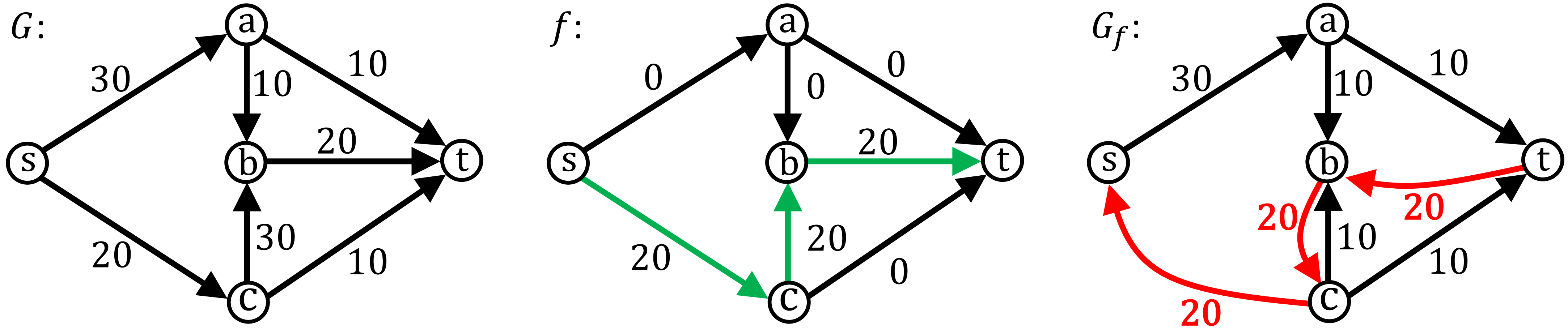


Algorithm Overview

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.

Remove edges with 0 capacity in residual graph. Can't use them anyway.

Max Flow Algorithm

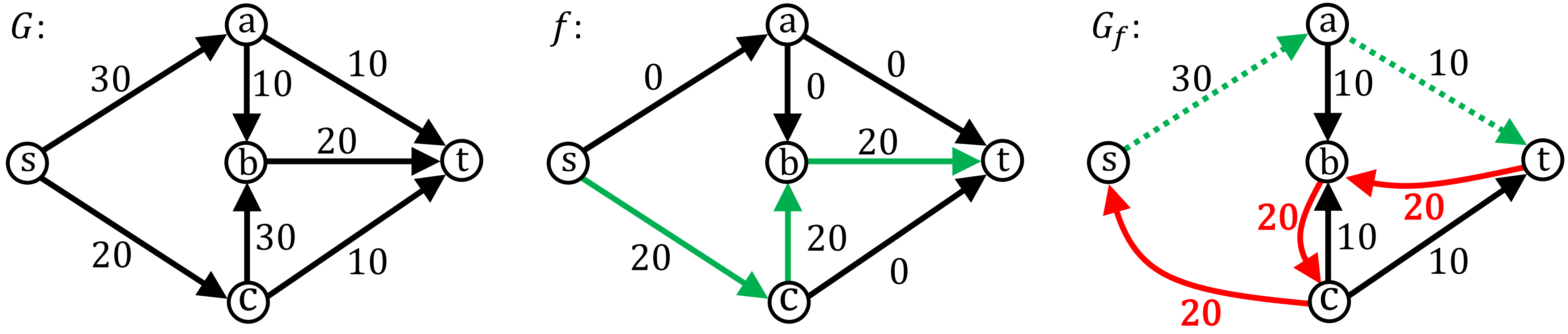


Algorithm Overview

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.

Remove edges with 0 capacity in residual graph. Can't use them anyway.

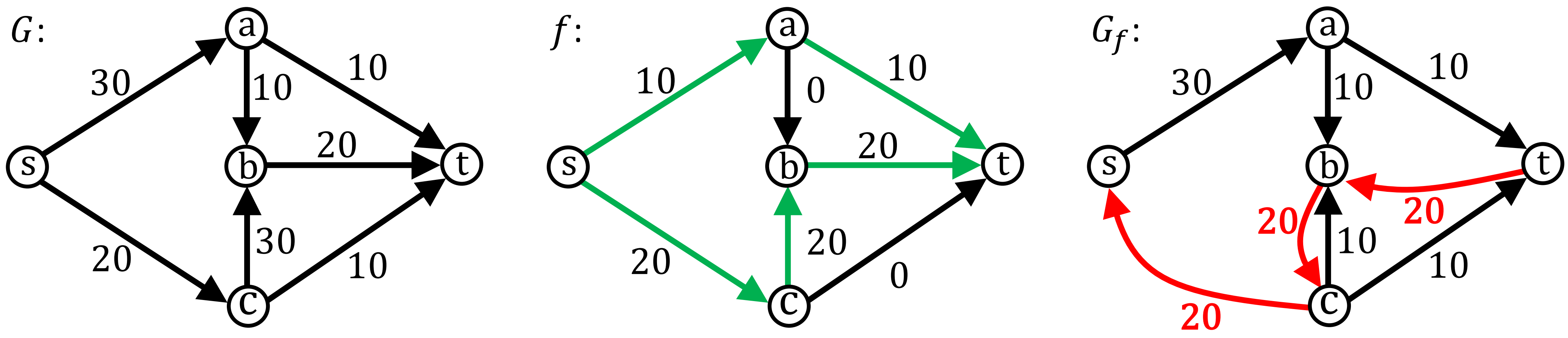
Max Flow Algorithm



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.

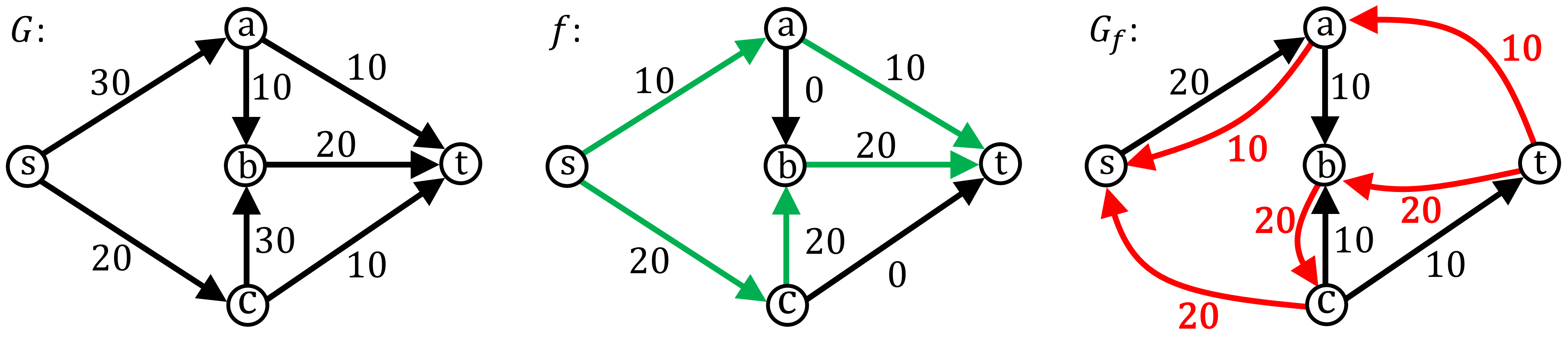
Max Flow Algorithm



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.

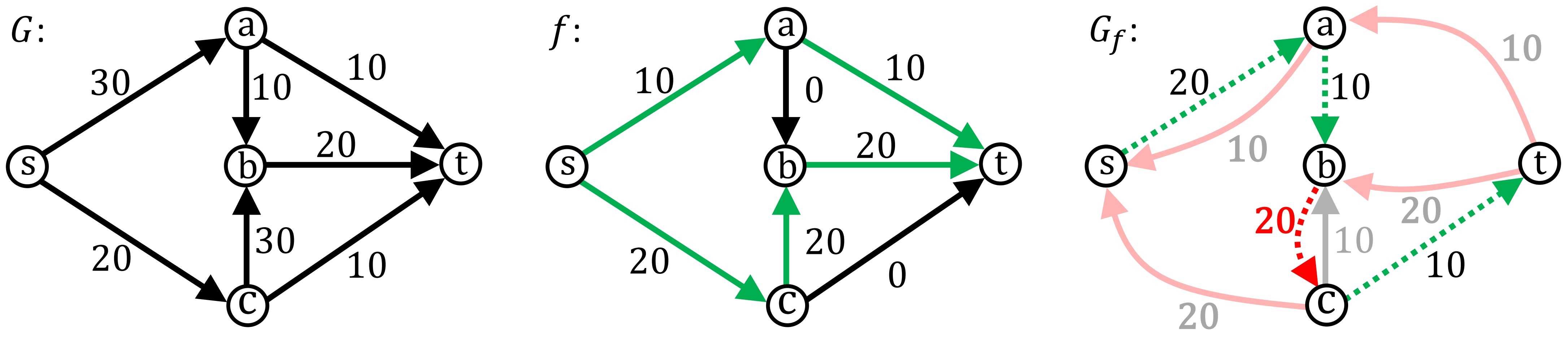
Max Flow Algorithm



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.

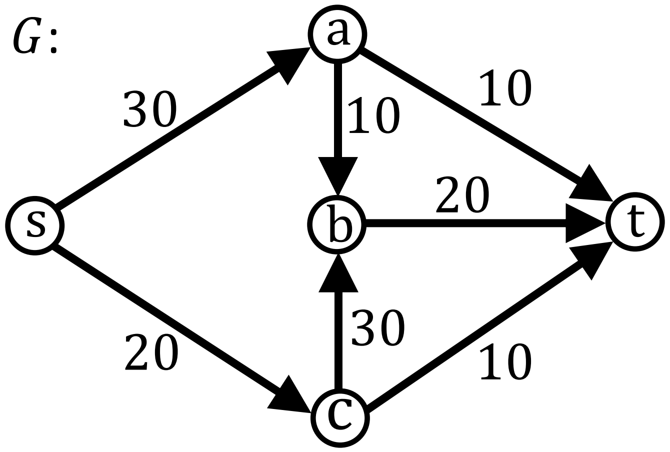
Max Flow Algorithm

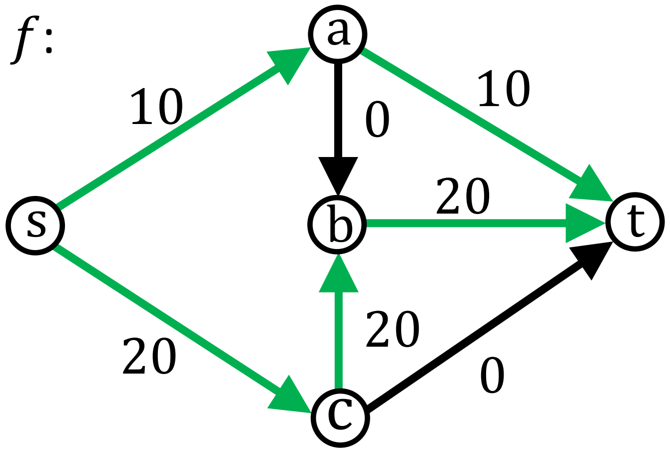


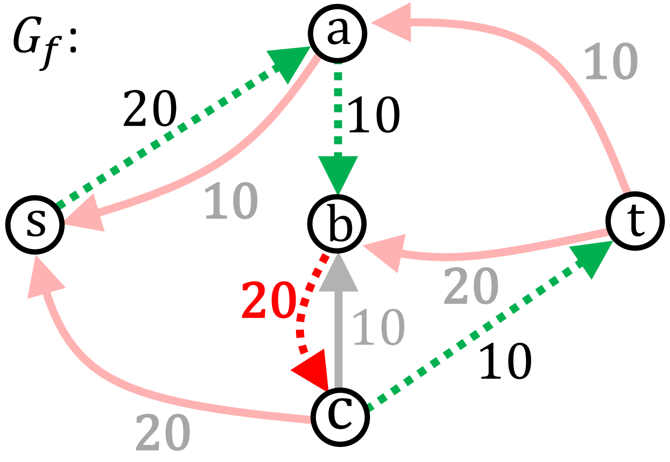
Algorithm Overview

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.

Max Flow Algorithm





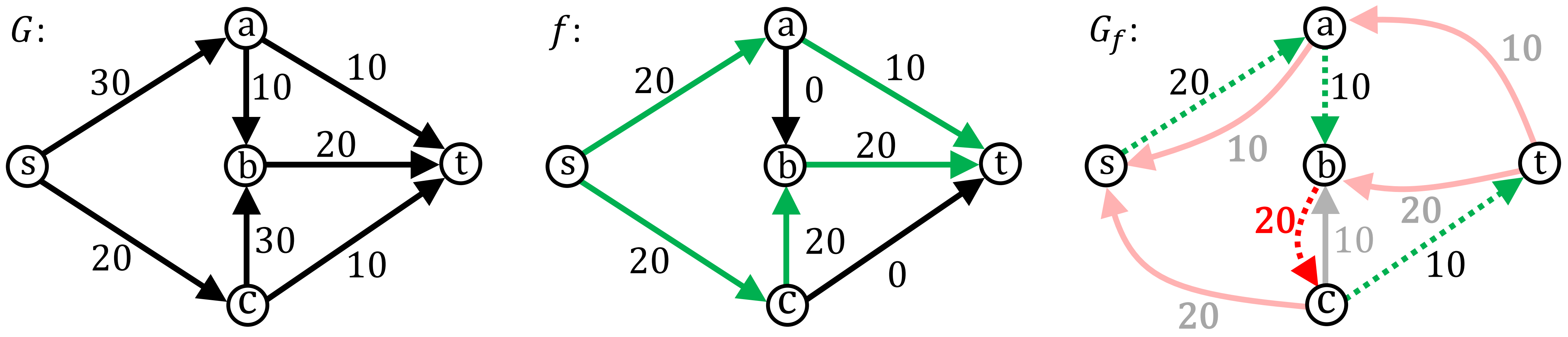


Algorithm Overview

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.

Pushing flow on a path:

Max Flow Algorithm



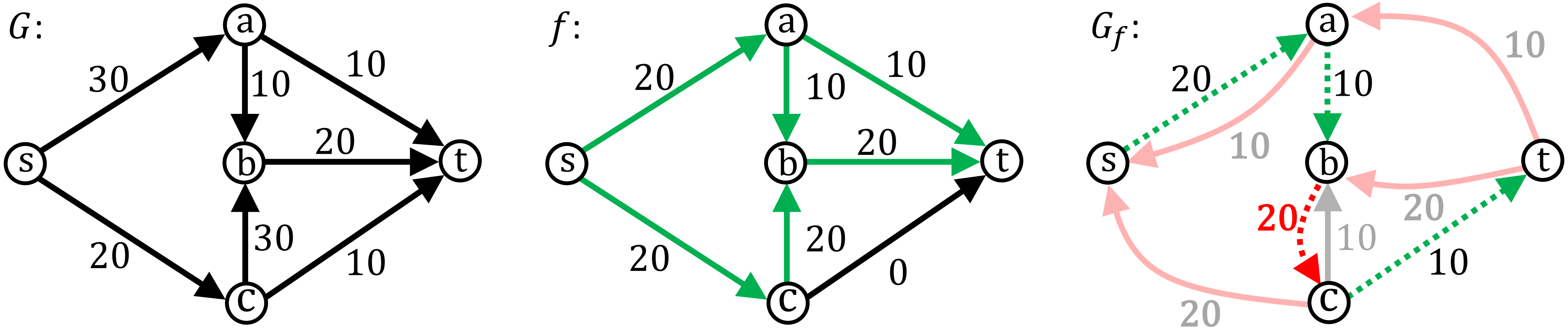
Algorithm Overview

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.

Pushing flow on a path:

1. If edge is a normal edge, increase flow.

Max Flow Algorithm



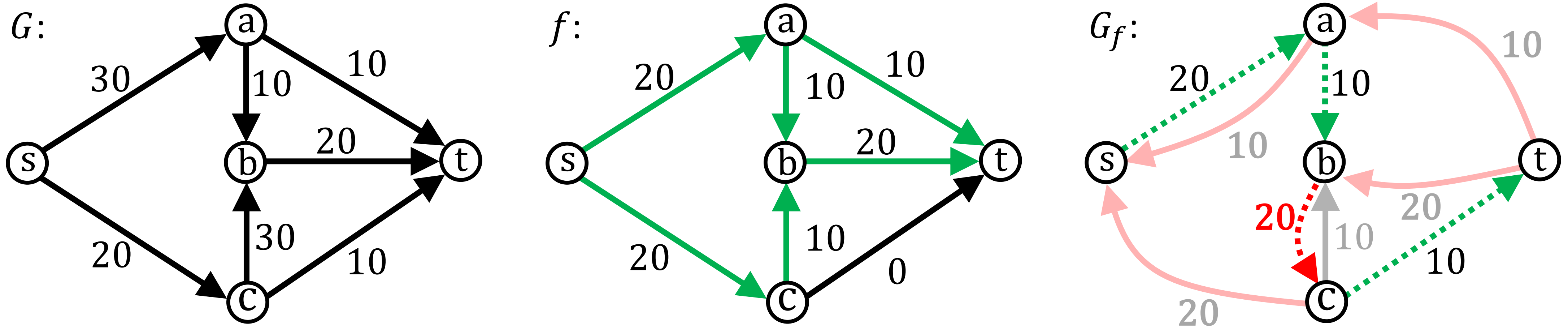
Algorithm Overview

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.

Pushing flow on a path:

1. If edge is a normal edge, increase flow.

Max Flow Algorithm



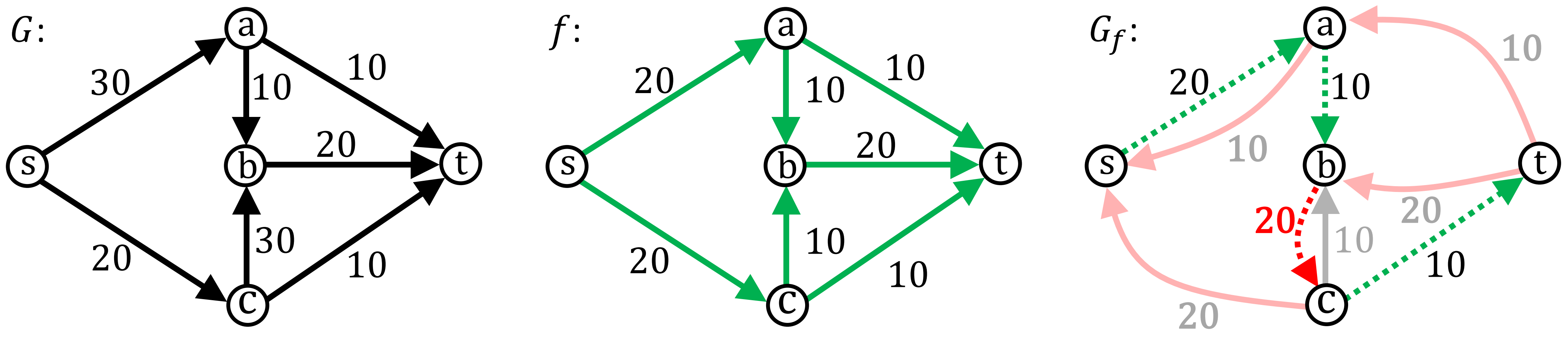
Algorithm Overview

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.

Pushing flow on a path:

1. If edge is a normal edge, increase flow.
2. If edge is a back edge, decrease flow.

Max Flow Algorithm



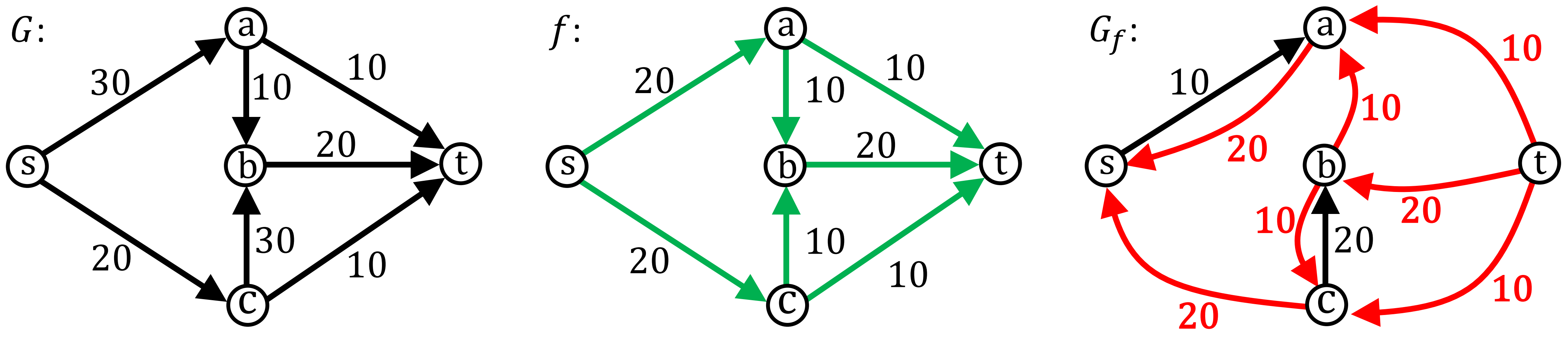
Algorithm Overview

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.

Pushing flow on a path:

1. If edge is a normal edge, increase flow.
2. If edge is a back edge, decrease flow.

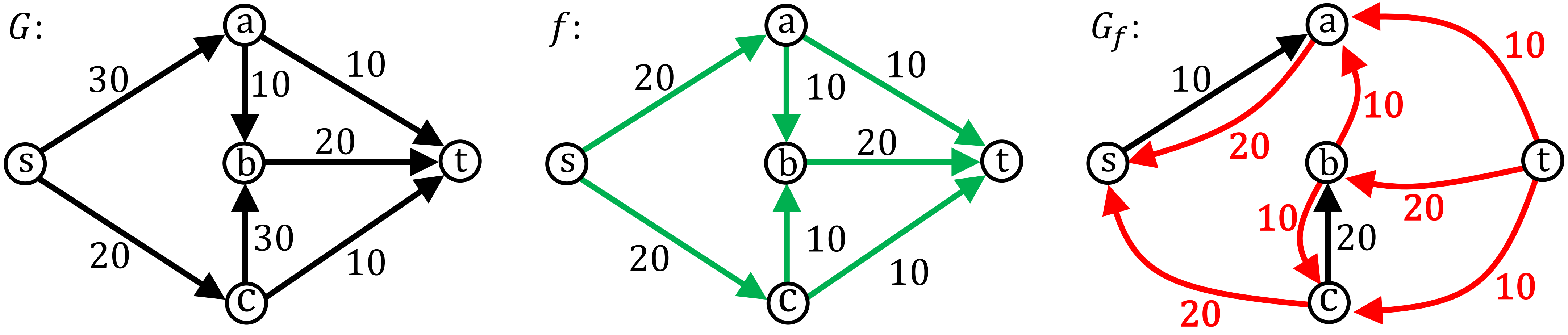
Max Flow Algorithm



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.

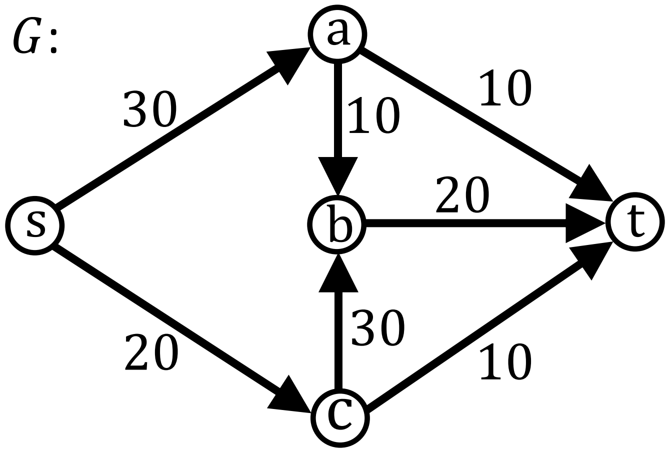
Max Flow Algorithm

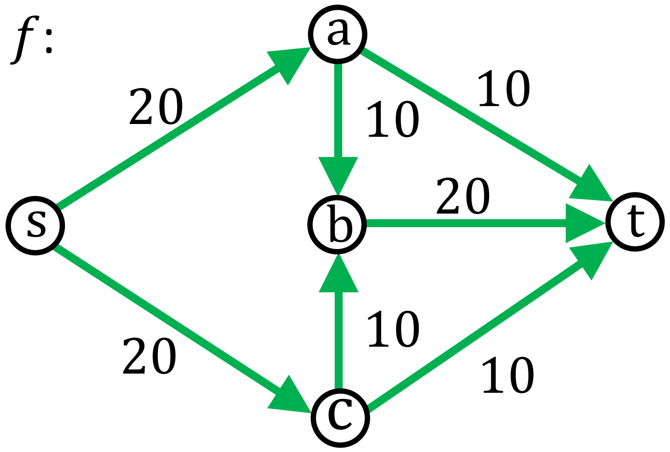


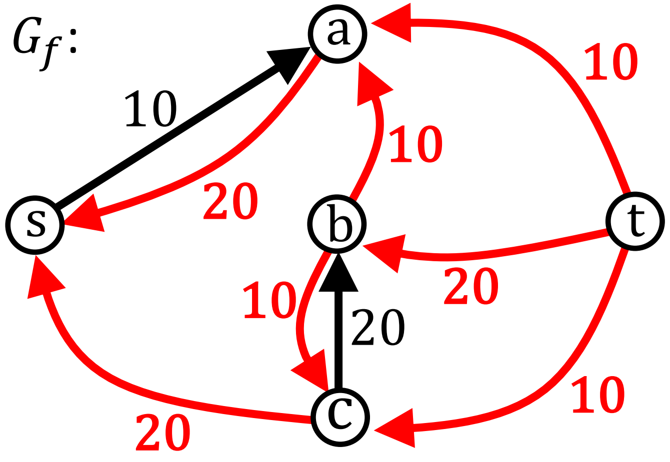
Algorithm Overview

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.

Max Flow Algorithm

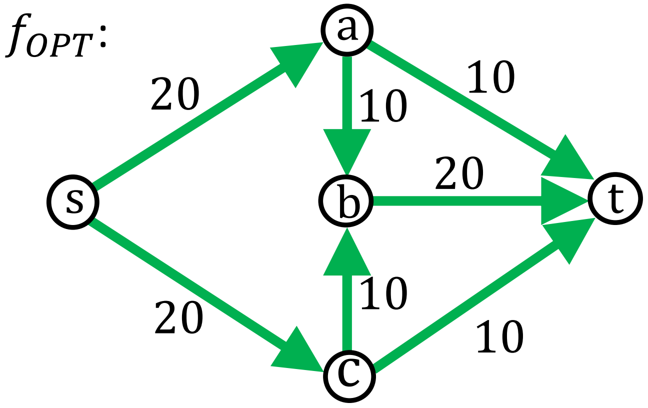




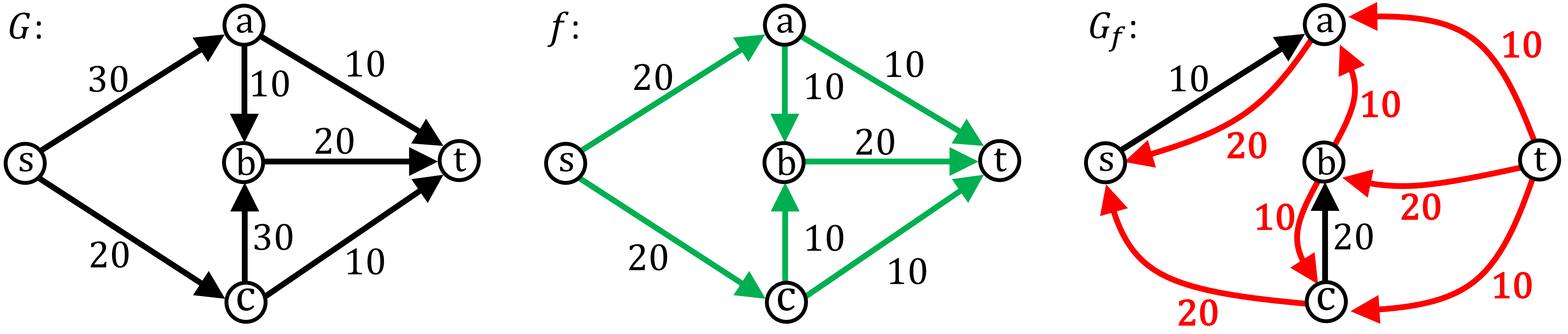


Algorithm Overview

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.



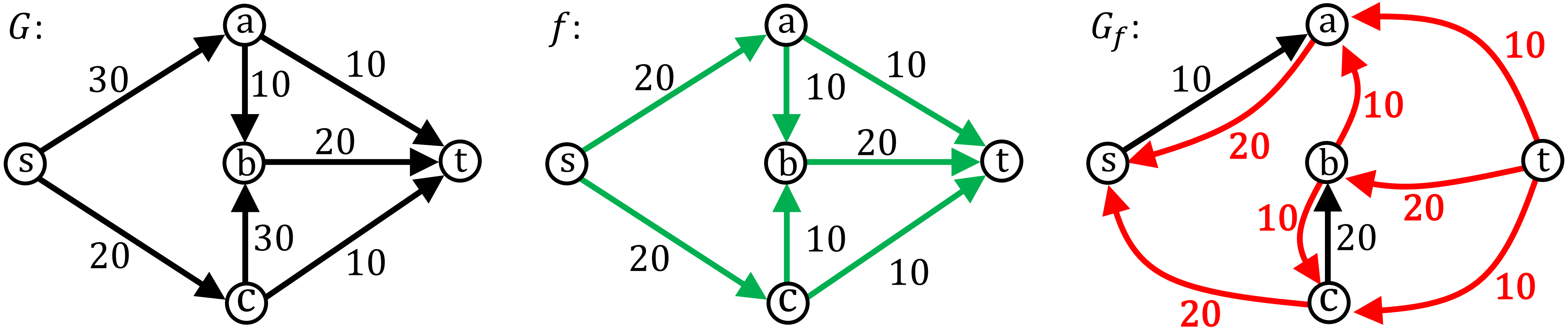
Ford-Fulkerson




```
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
   while s-t path in
      P = simple s-t
      f'= augment(f,
      f = f'
      Gf = Gf'
   return f
```

```
augment(f, P)
      bottleneck(P,f)
      each edge (u, v) in P
      f (u, v) is a back edge
         f((v, u)) -= b
      se
         f((u, v)) += b
   return f
```

Need to show:

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

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

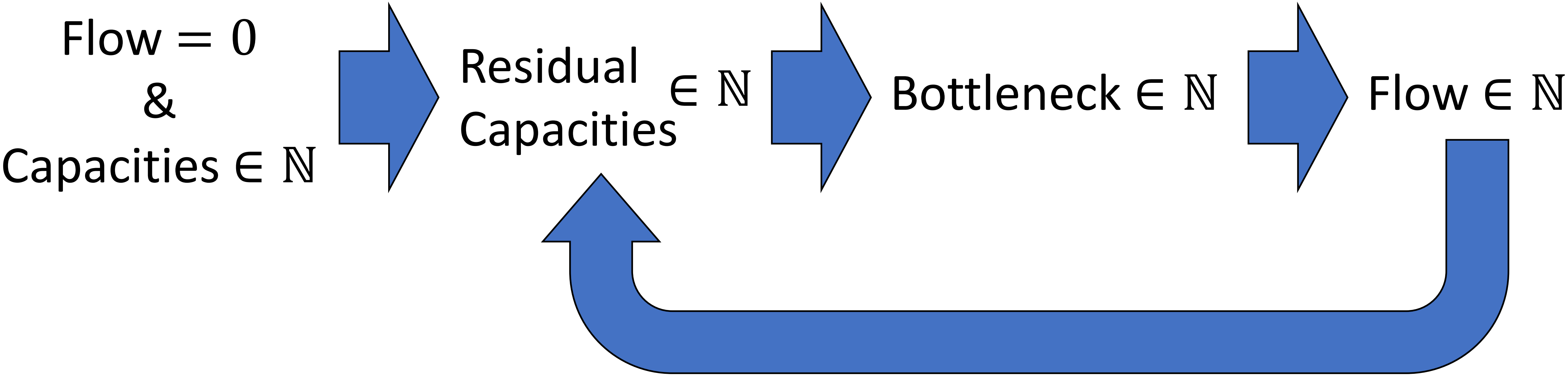
Claims:

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

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.

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.

Bottleneck $\in \mathbb{N}$

Ford-Fulkerson

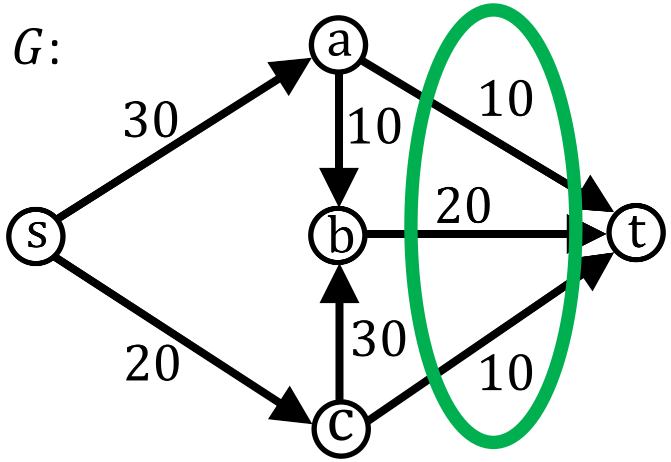
Claims:

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.



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.