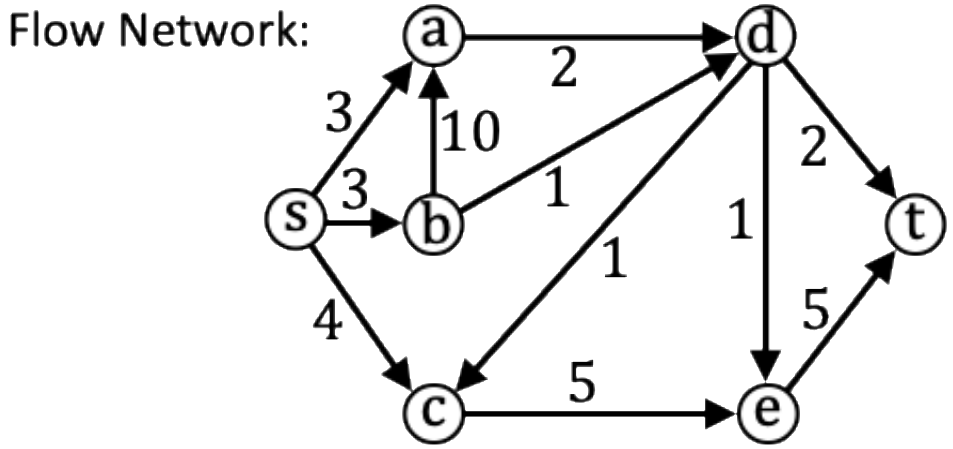
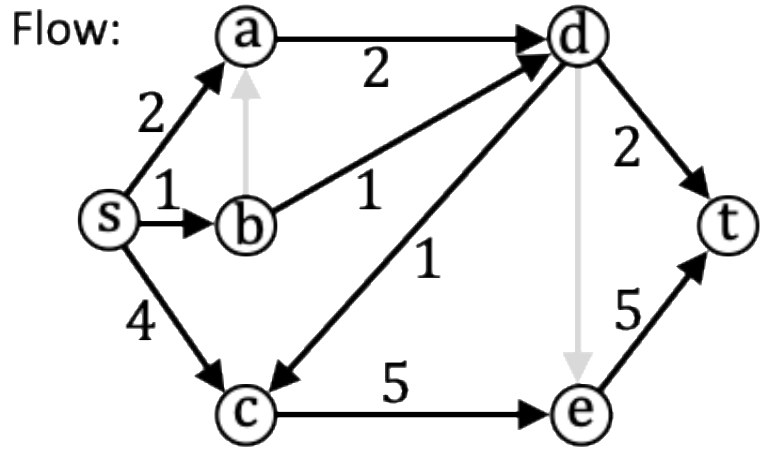
Flow Networks
CSCI 432

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



Flow Network:

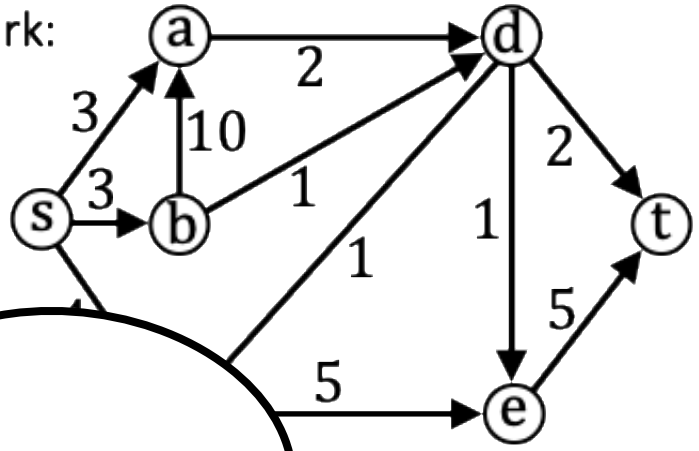
- Directed-edge graph, $G = (V, E)$.
- Non-negative edge capacity, c_e.
- Single source, s, without input edges.
- Single sink, t, without output edges.



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

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

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



Flow Network:

- Direct
- Non-
-
-

An s -

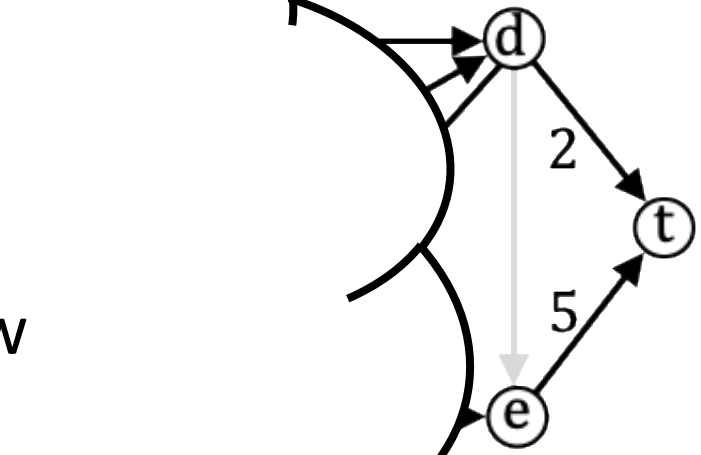
- 0
- $\sum_{e \epsilon \text{input}}$ (conservation of flow: [illegible] goes into a node has to come out, except s and t")
- Value of flow = $val(f) = \sum_{e \epsilon \text{output}(s)} f(e) = \sum_{e \epsilon \text{input}(t)} f(e)$

How do we find max flows?

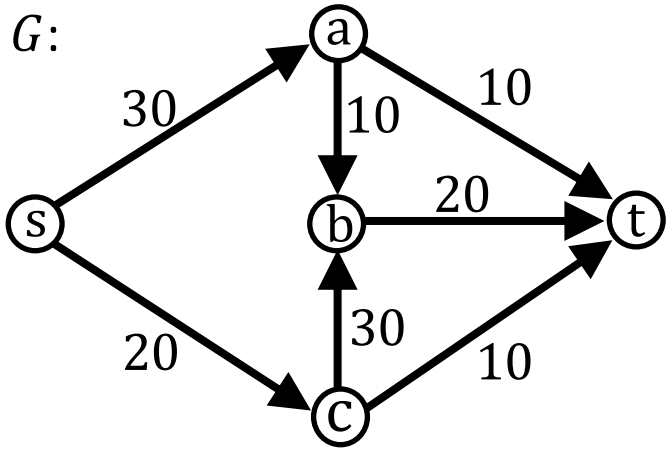
We'll get to that.

Assumptions for now:

1. We can find max flows for valid flow networks.
2. If the capacities of a flow network are integer-valued, the max flow will be too.



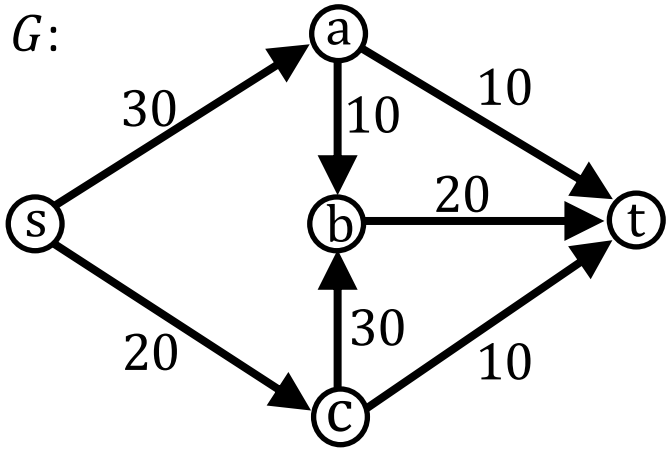
Max Flow Algorithm



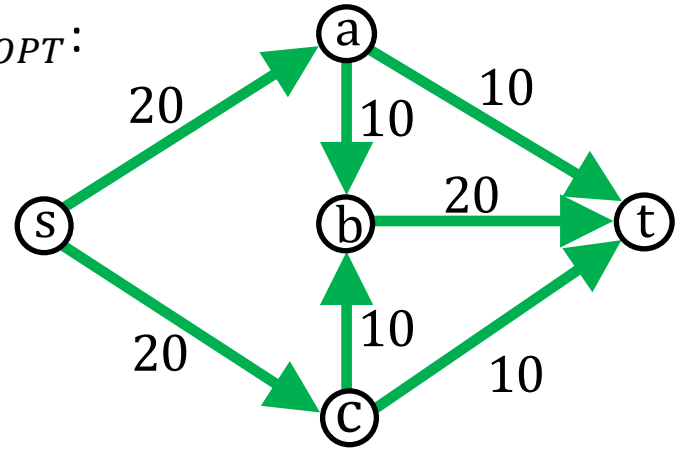
Max Flow?

Max Flow Algorithm

G:

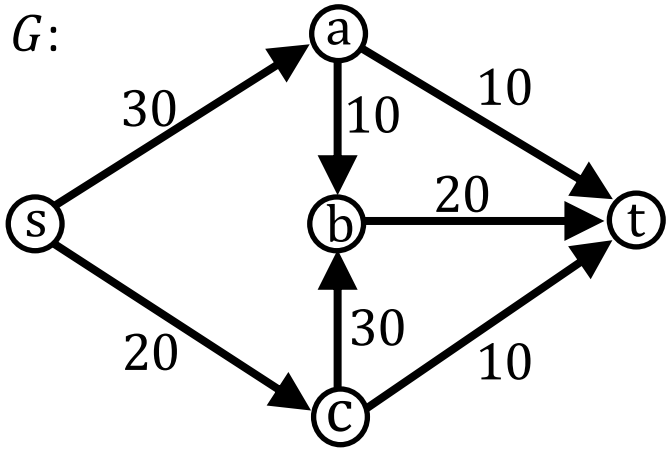


f_{OPT}:

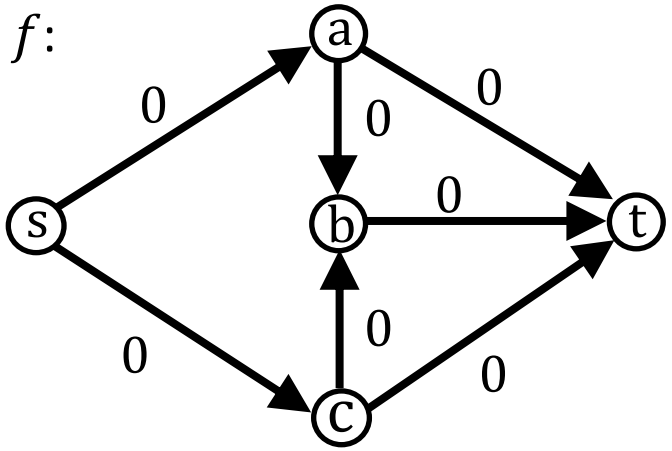


Max Flow Algorithm

G:

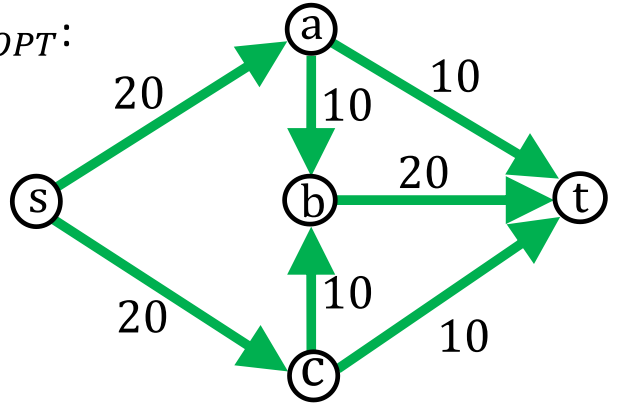


f:

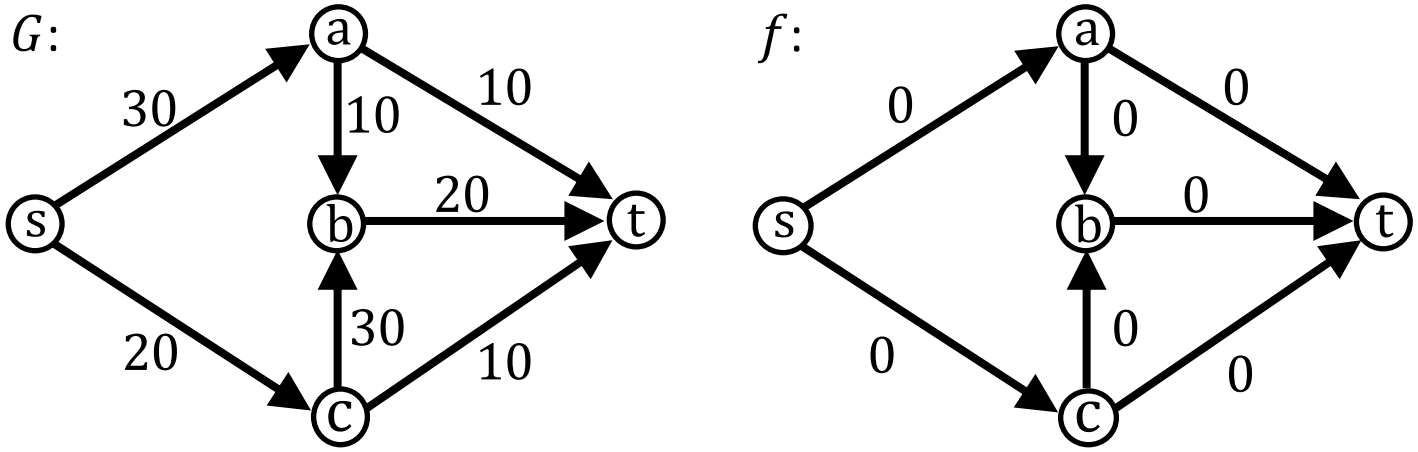


Ideas?

f_{OPT}:

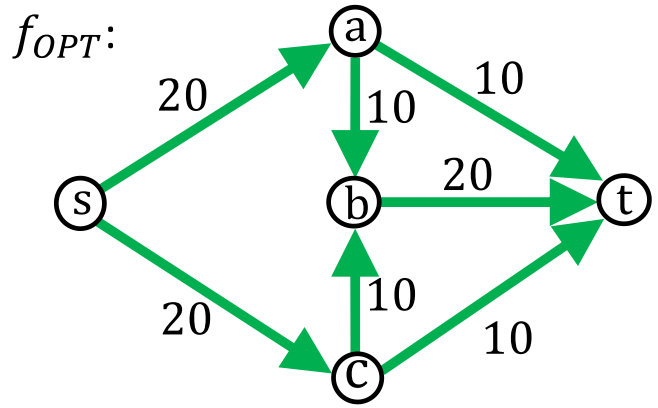


Max Flow Algorithm



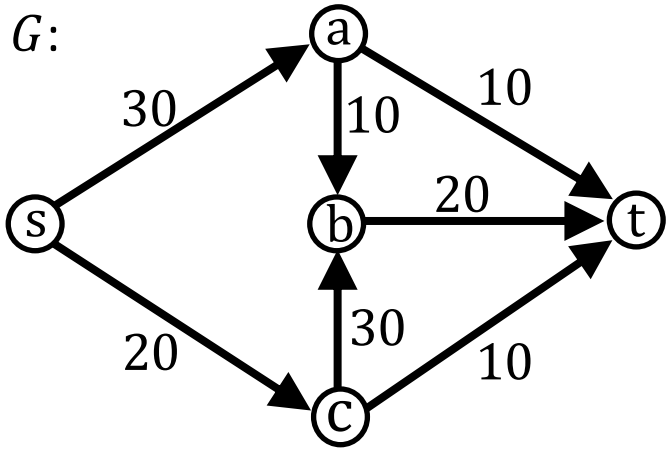
Ideas?

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

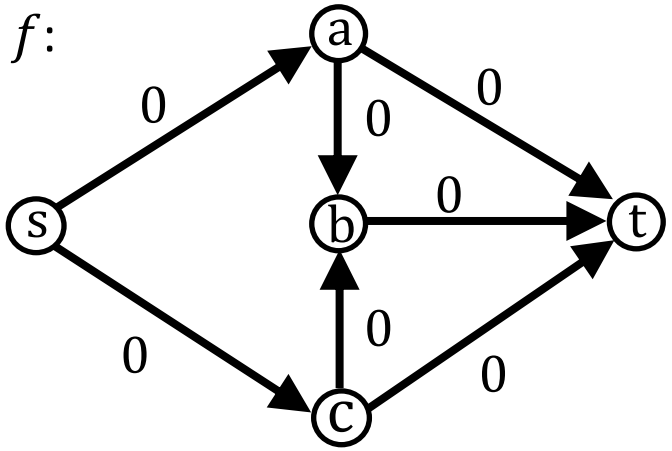


Max Flow Algorithm

G:



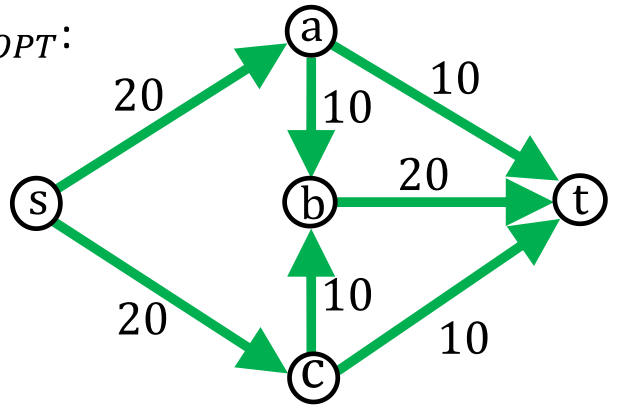
f:



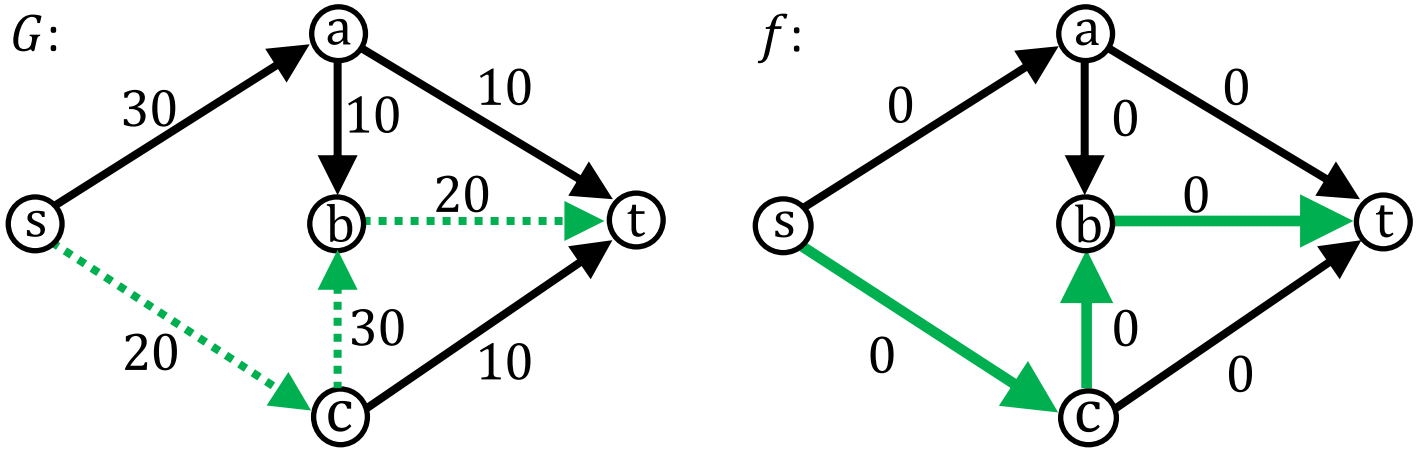
Ideas?

1. Select a path P.

f_{OPT}:



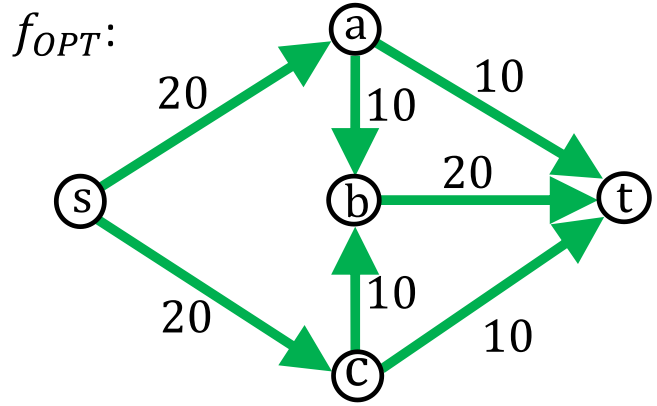
Max Flow Algorithm



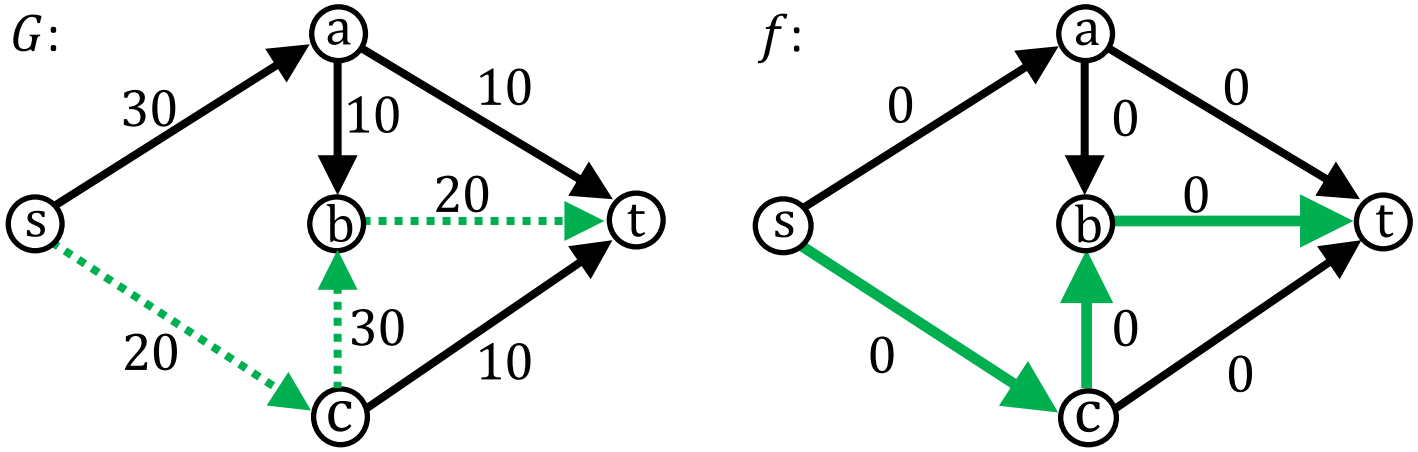
Ideas?

1. Select a path P.

How much flow should we push?



Max Flow Algorithm

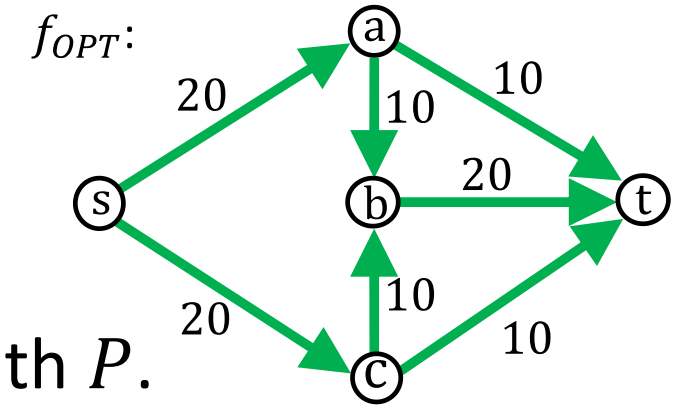


Ideas?

1. Select a path P.

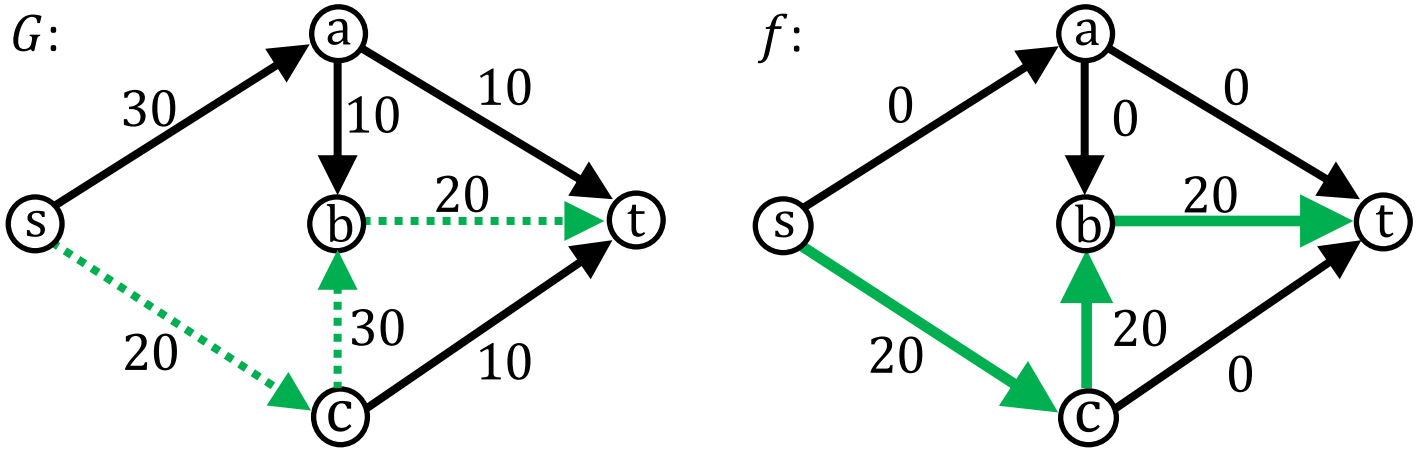
How much flow should we push?

As much as possible.



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

Max Flow Algorithm

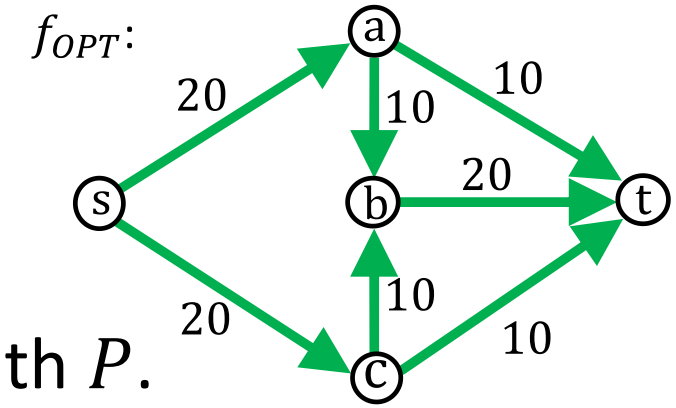


Ideas?

1. Select a path P.

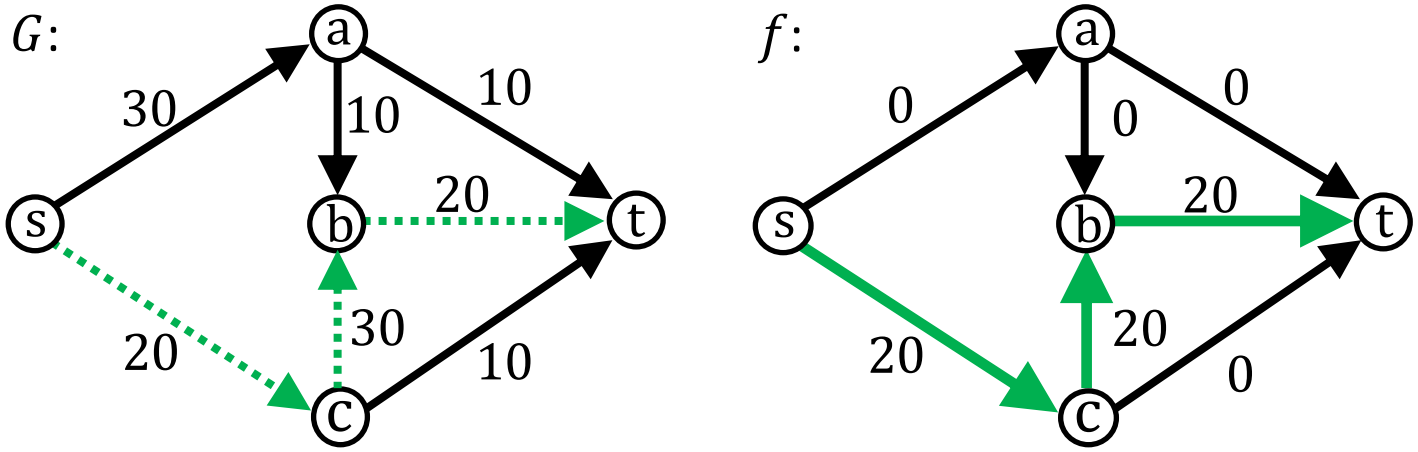
How much flow should we push?

As much as possible.



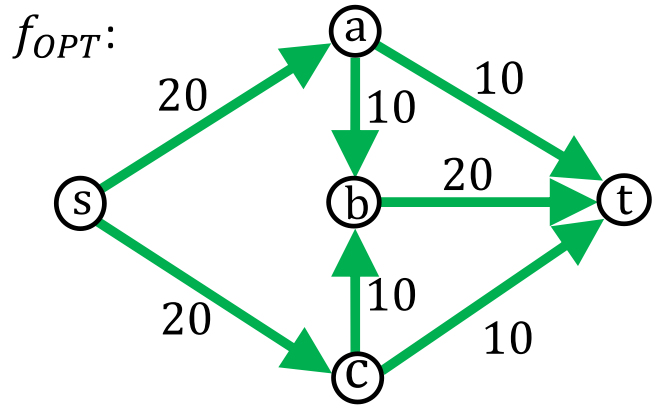
`bottleneck(P)` = minimum capacity on any edge in path P.

Max Flow Algorithm

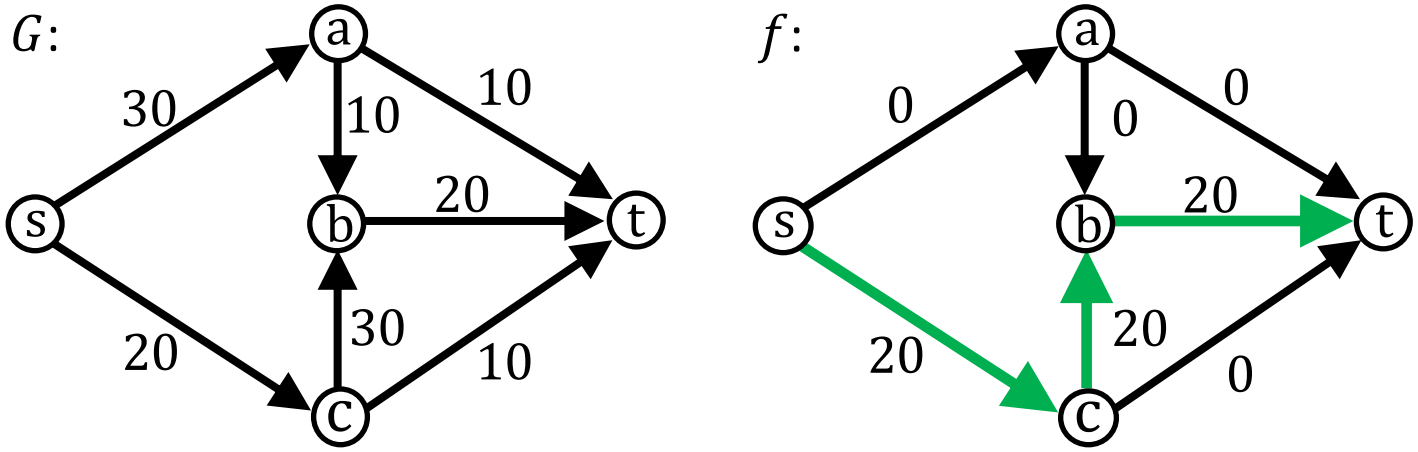


Ideas?

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



Max Flow Algorithm

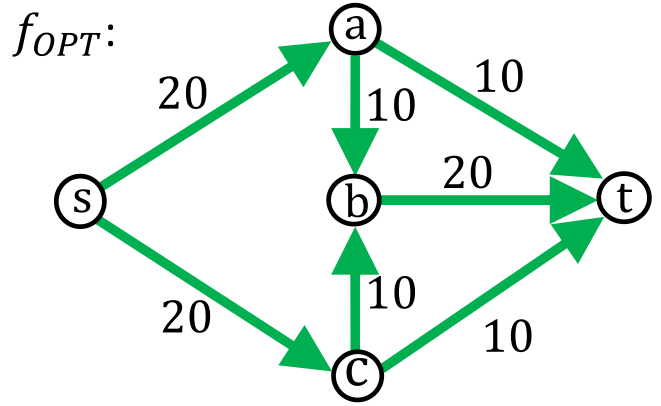


Ideas?

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

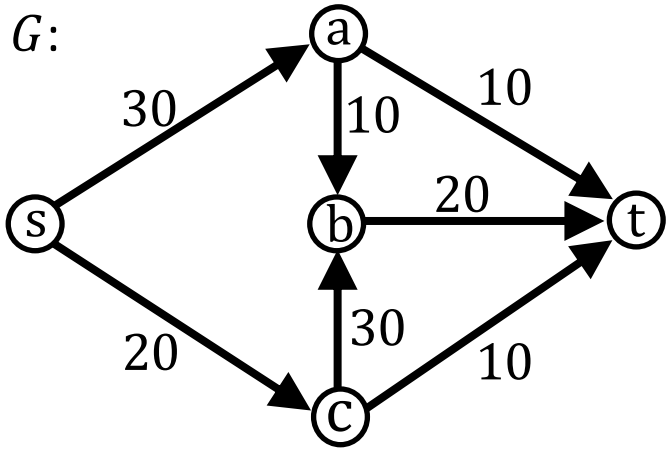
So far:

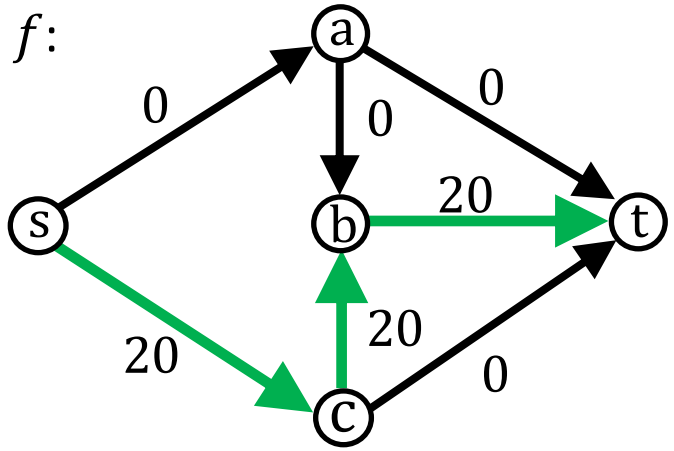
- **Guaranteed to meet conservation of flow constraints.**
- **Guaranteed to meet capacity constraints.**

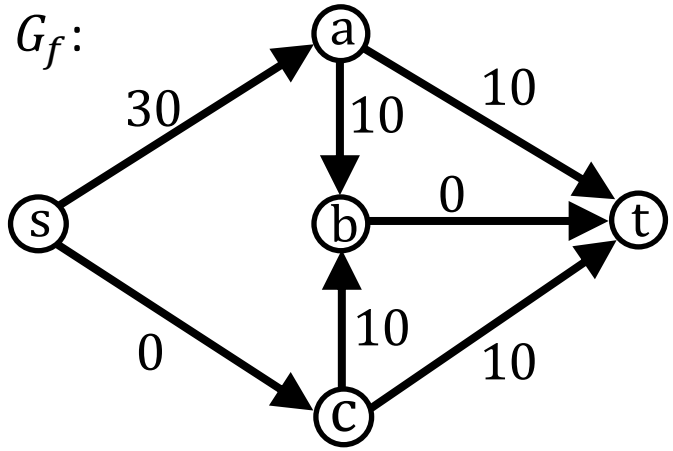


Max Flow Algorithm

Residual Capacity

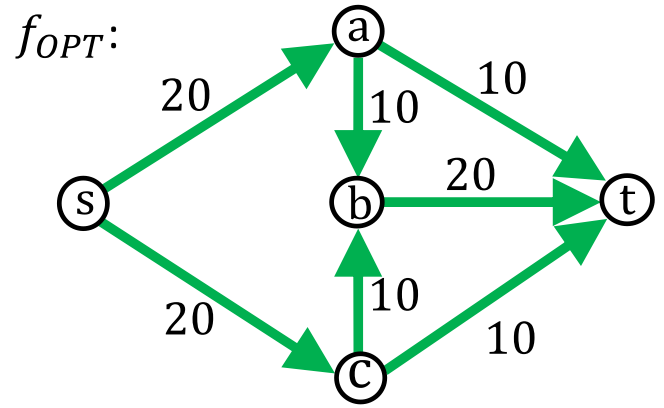




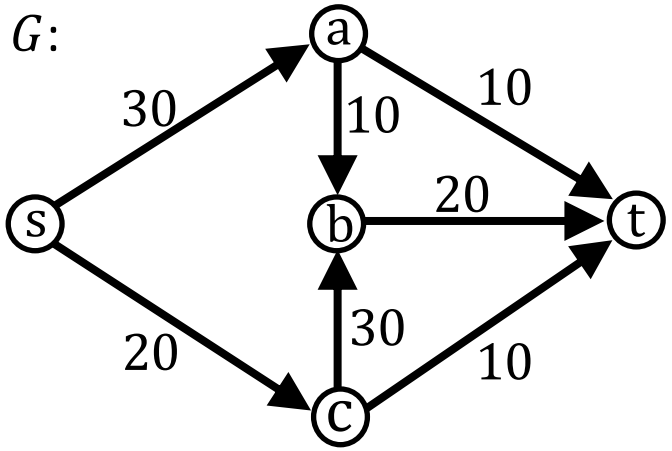


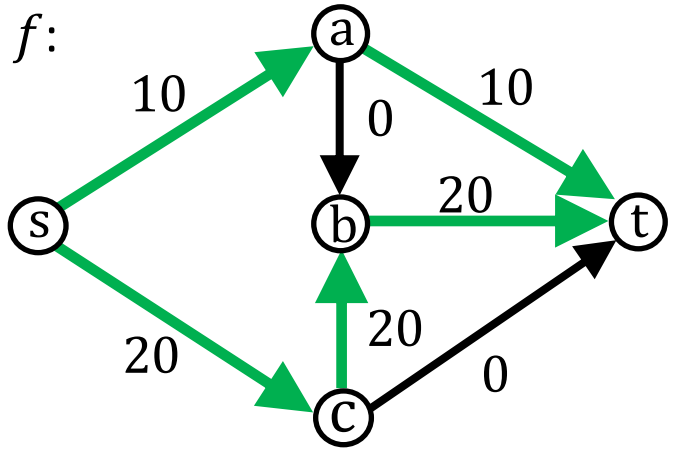
Ideas?

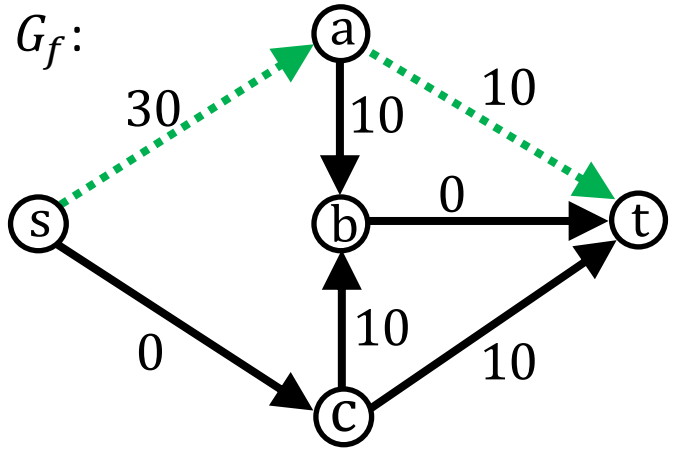
1. Select a path P.
2. Push bottleneck(P) flow on P.



Max Flow Algorithm

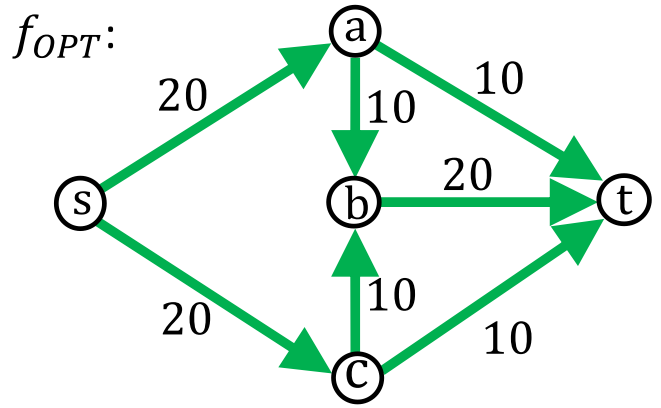




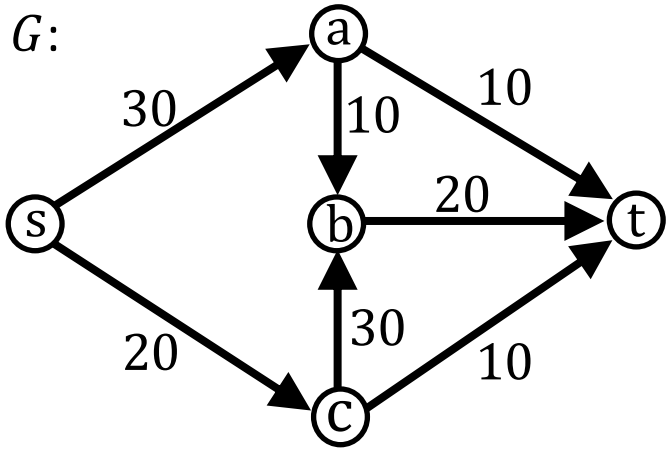


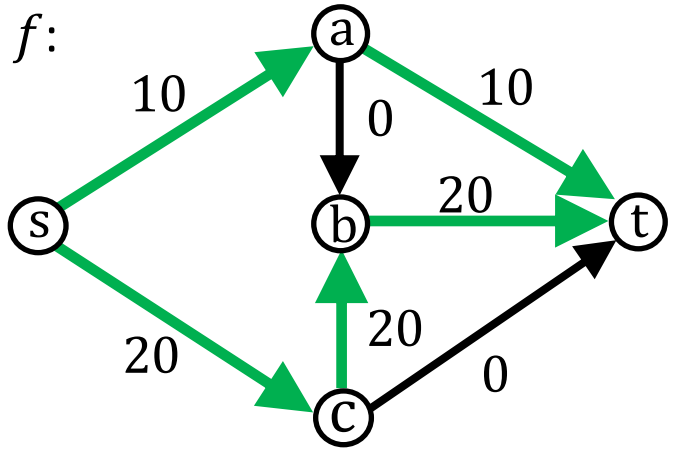
Ideas?

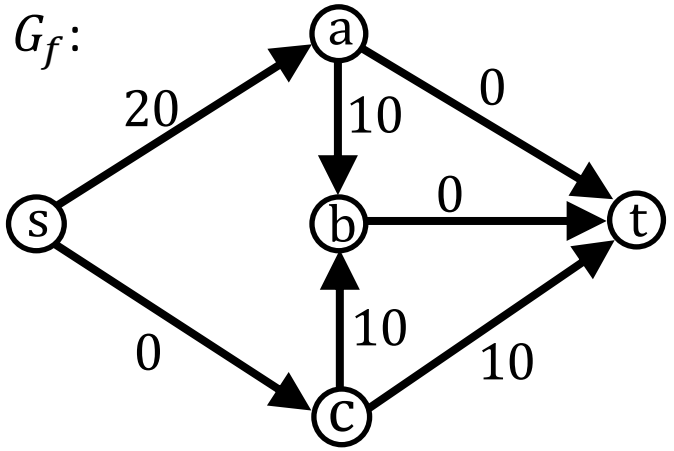
1. Select a path P.
2. Push bottleneck(P) flow on P.



Max Flow Algorithm

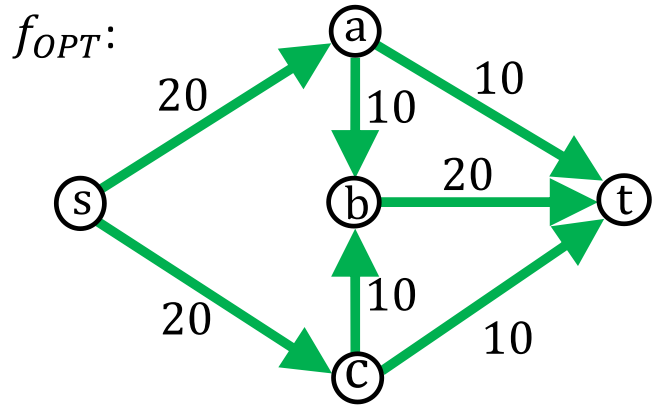






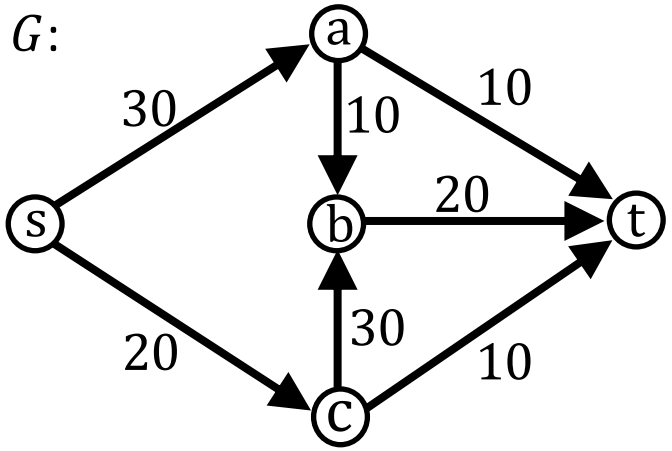
Ideas?

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

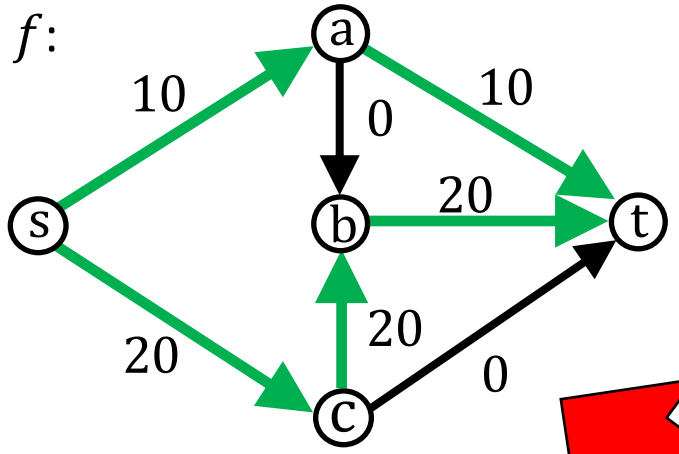


Max Flow Algorithm

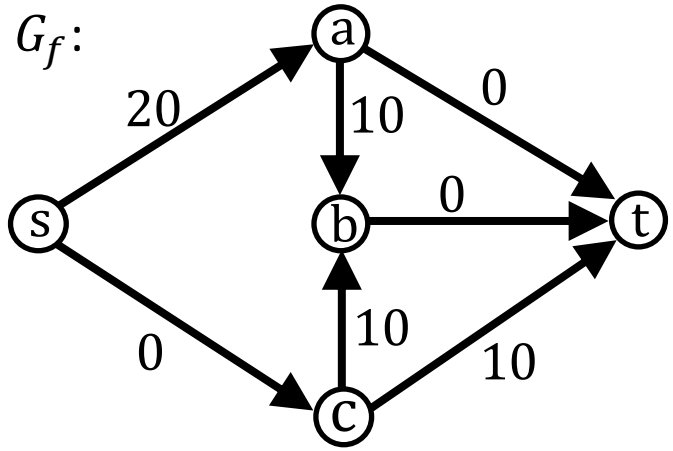
G:



f:

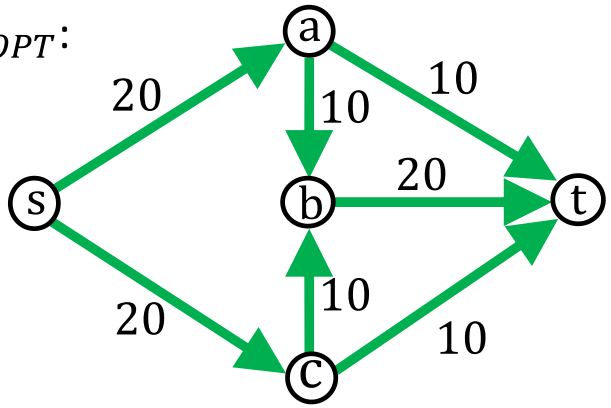


G_f:

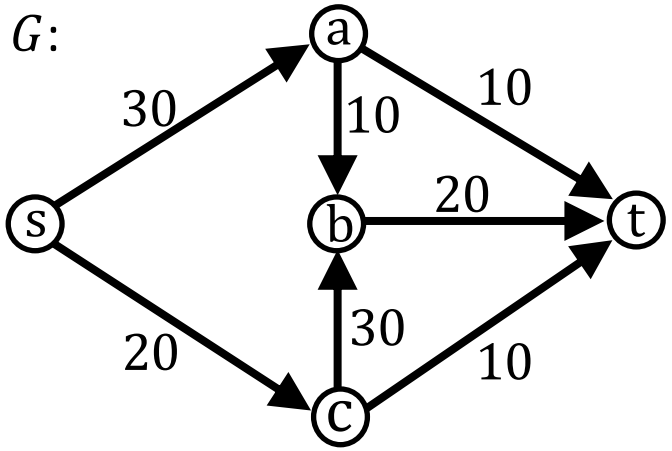


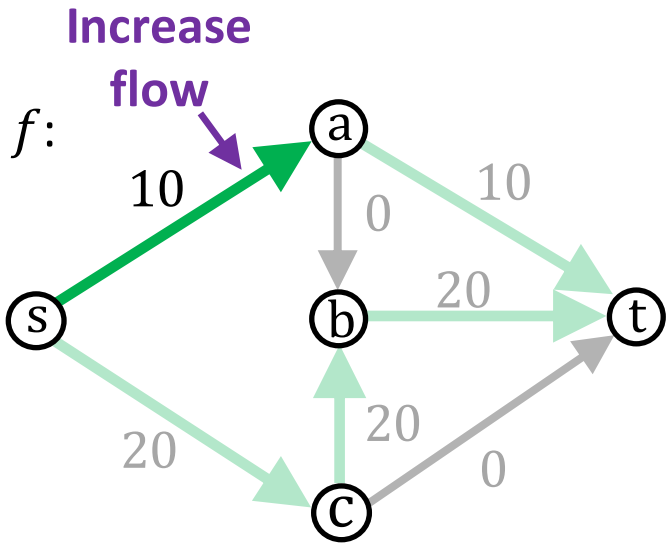
We need some way to reroute flow.

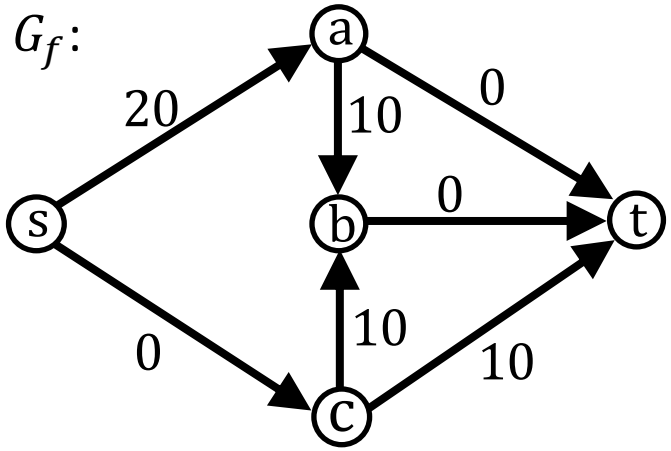
f_{OPT}:



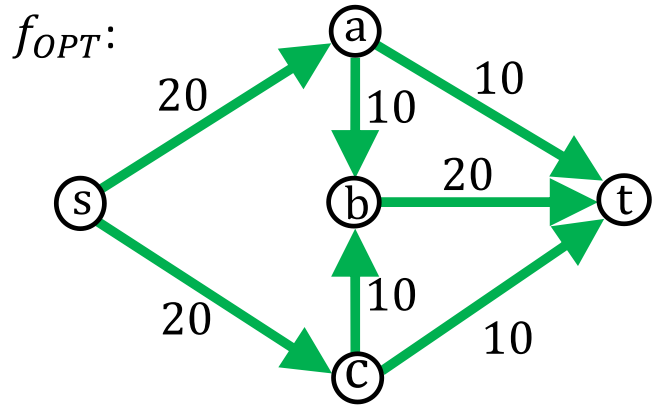
Max Flow Algorithm



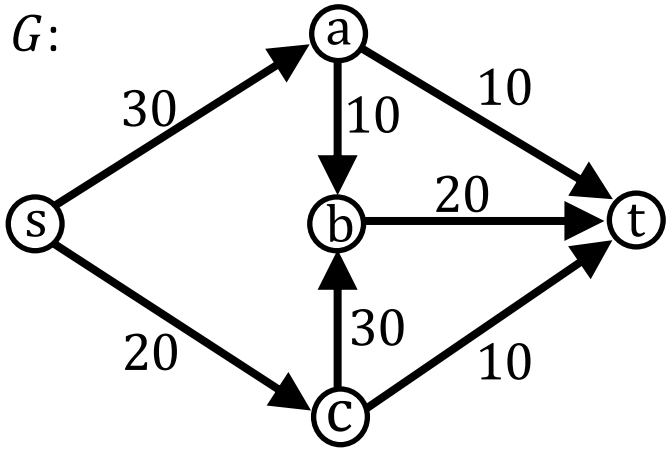


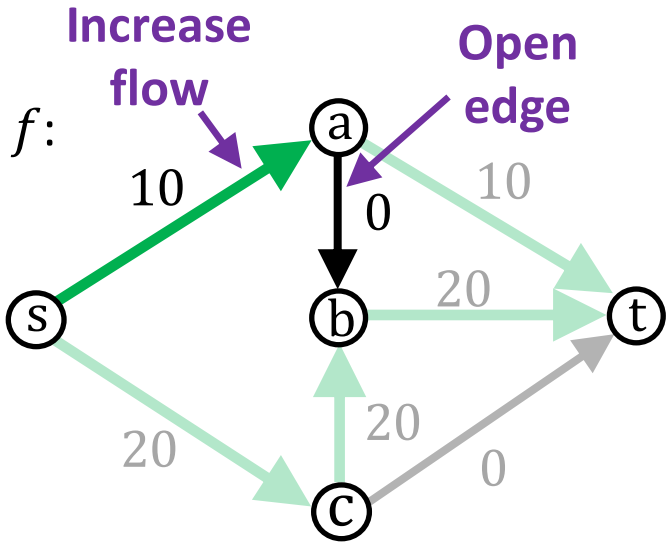


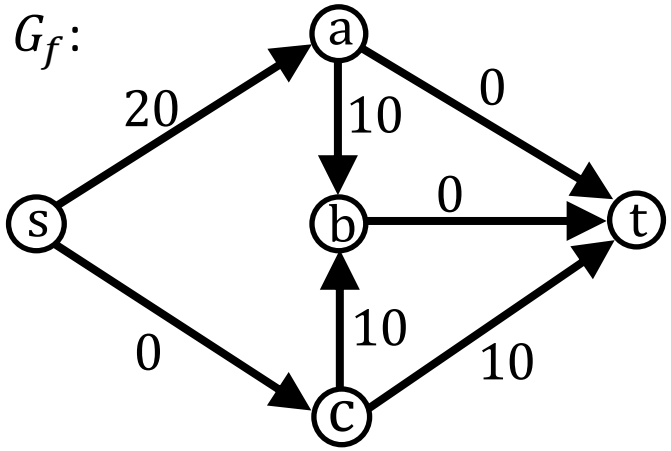
We need some way to reroute flow.



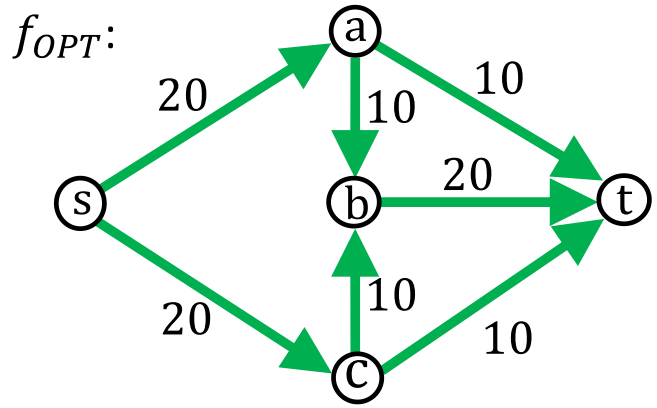
Max Flow Algorithm



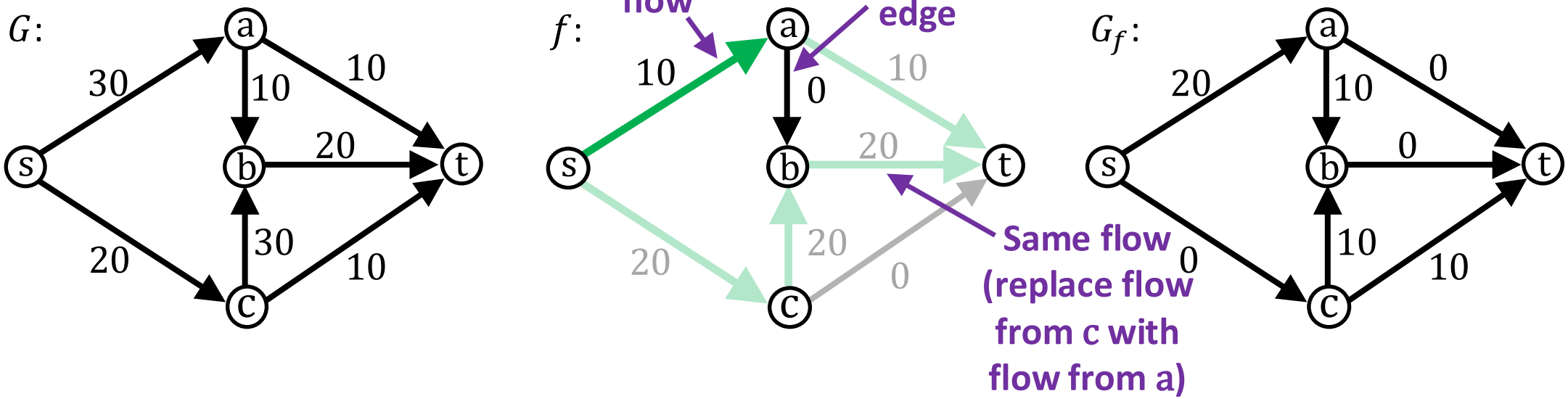




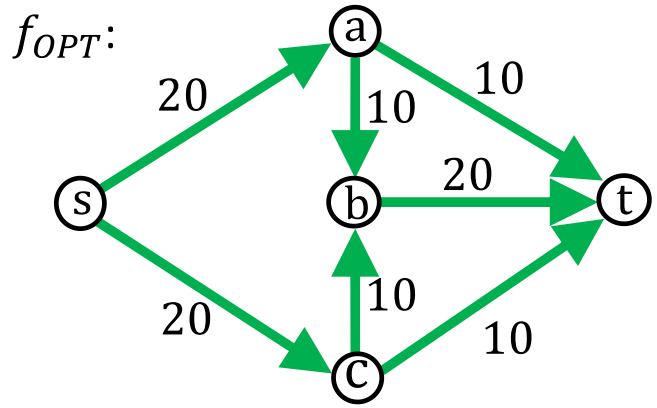
We need some way to reroute flow.



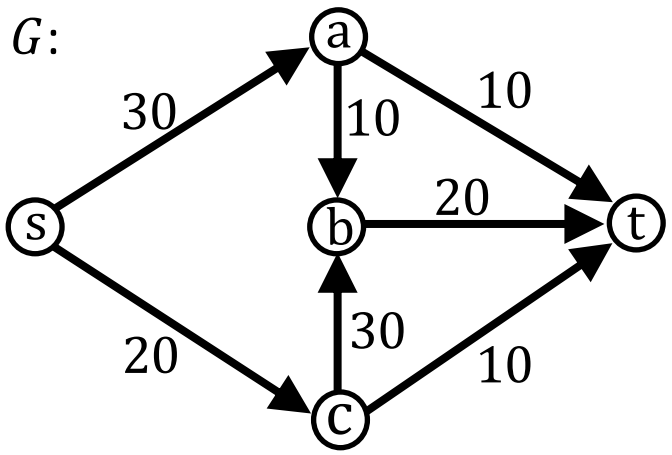
Max Flow Algorithm

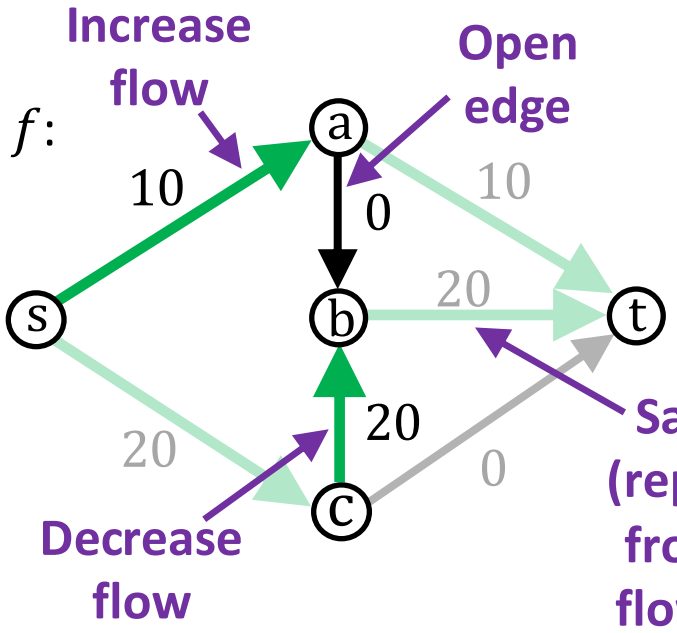


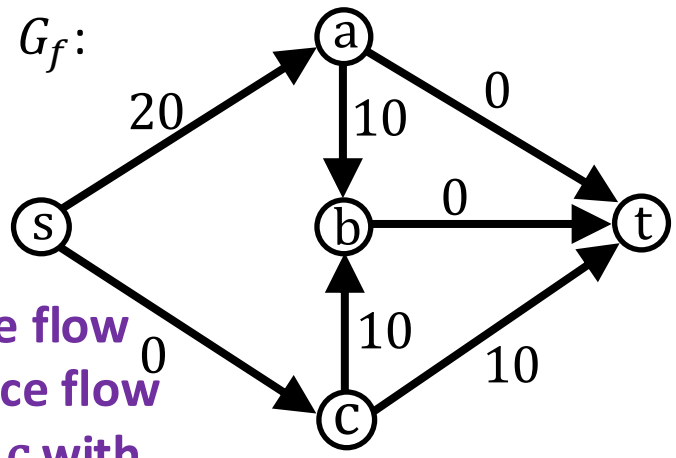
We need some way to reroute flow.



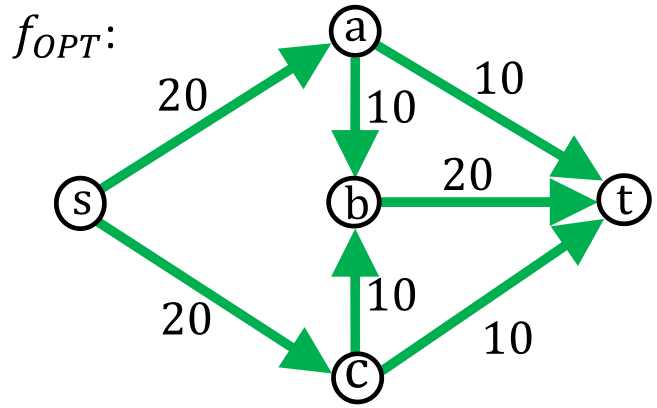
Max Flow Algorithm



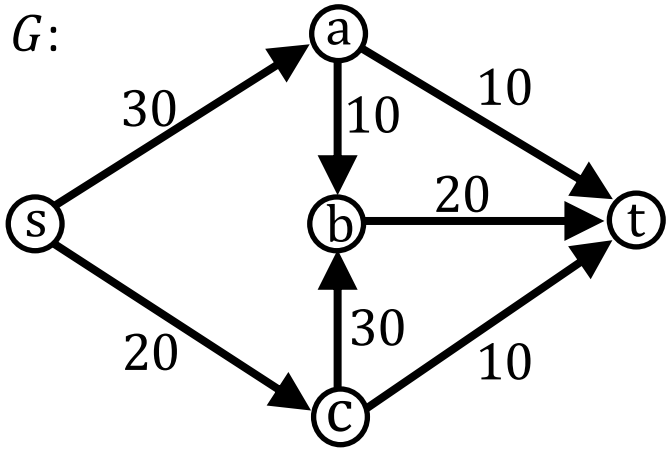


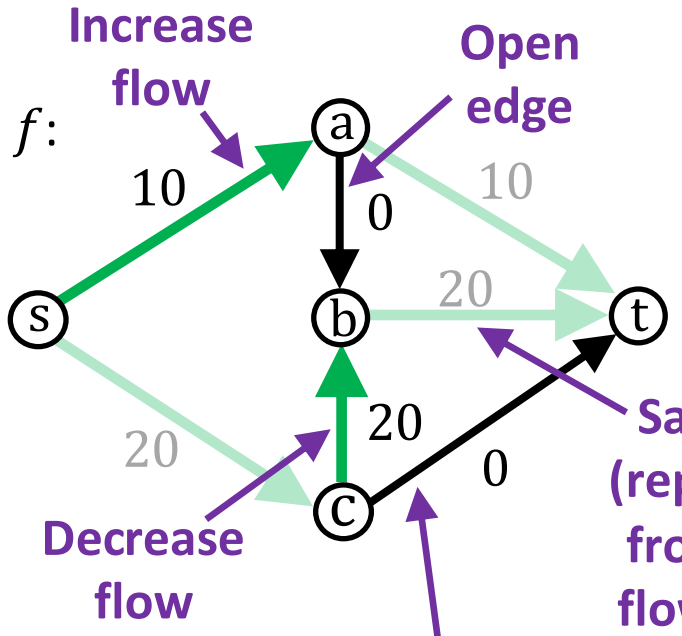


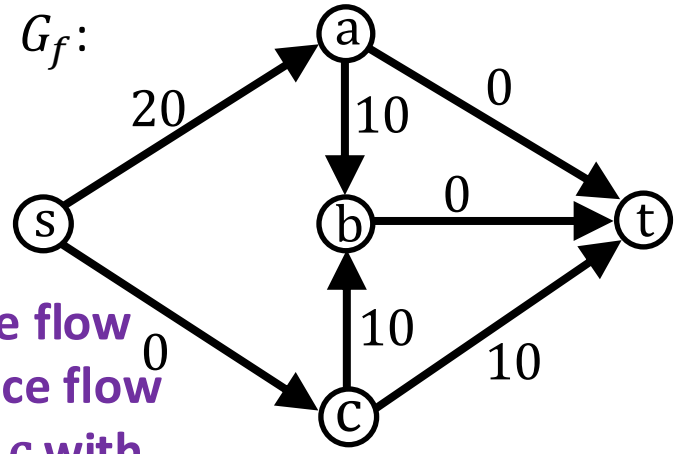
We need some way to reroute flow.



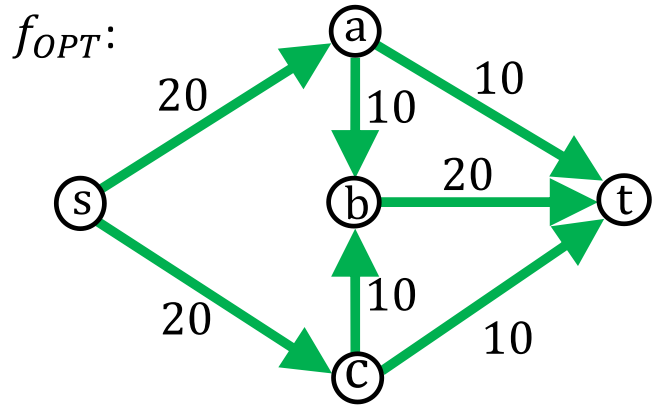
Max Flow Algorithm



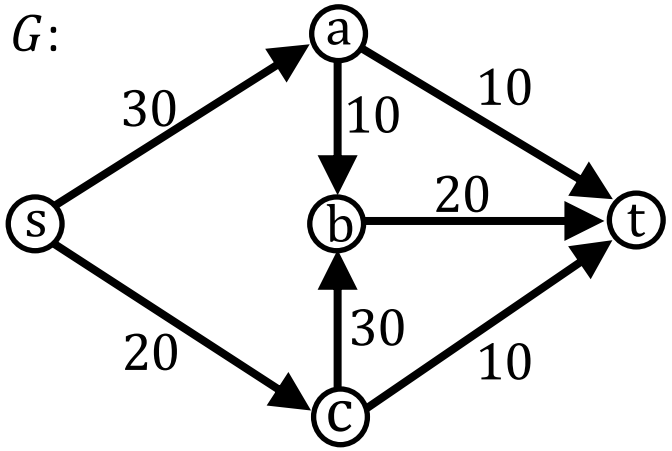


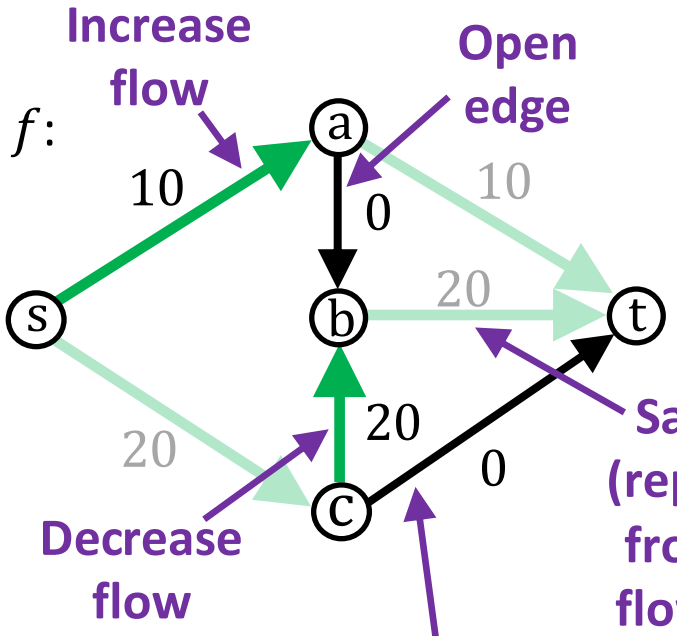


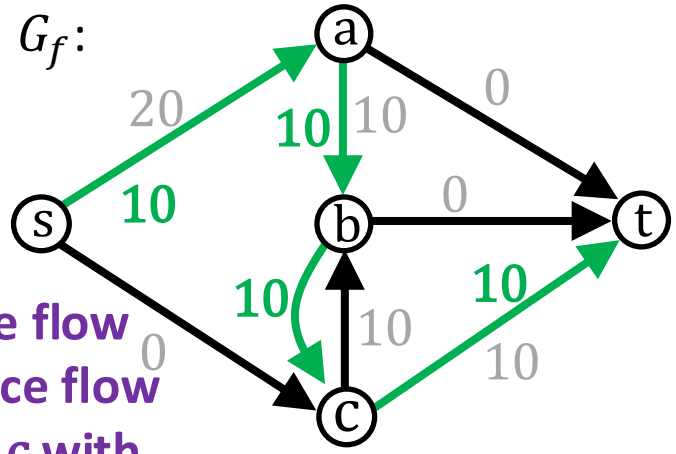
We need some way to reroute flow.



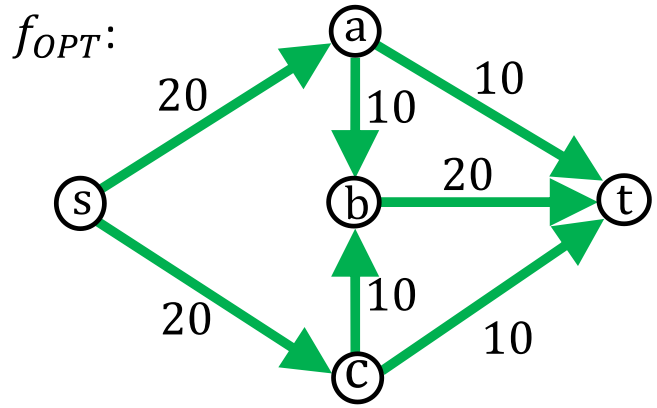
Max Flow Algorithm



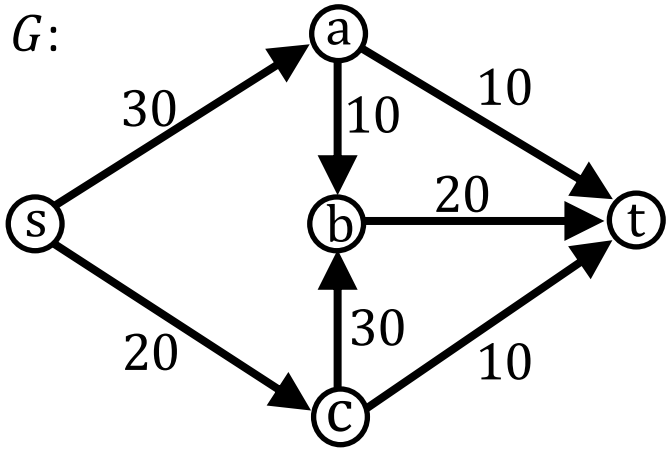


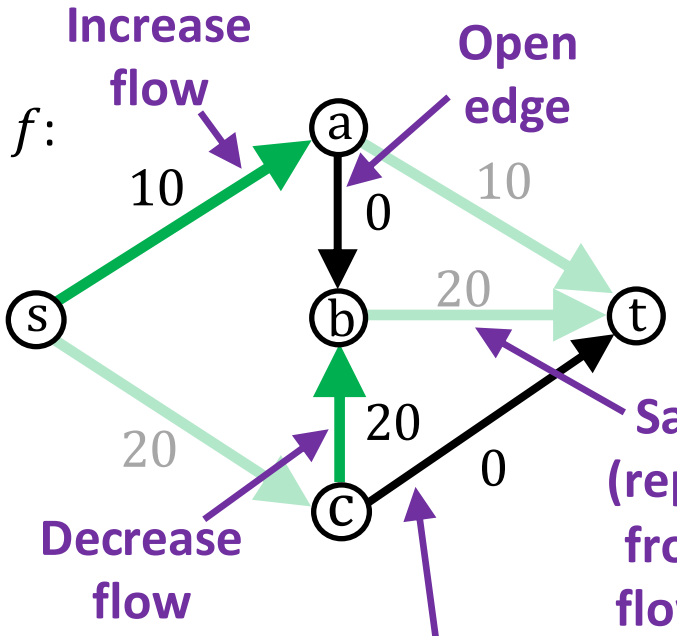


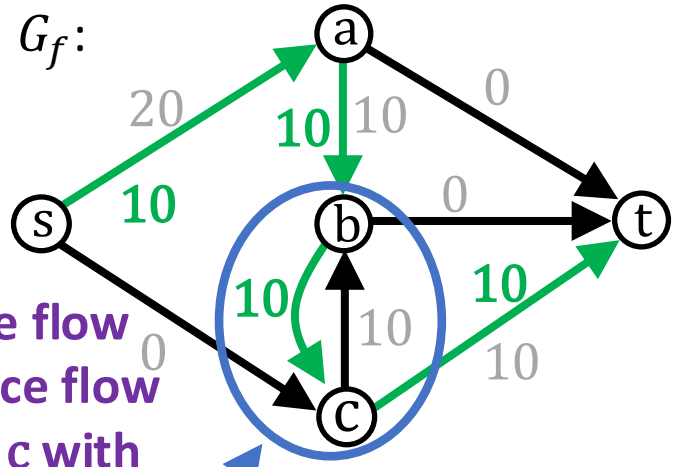
We need some way to reroute flow.



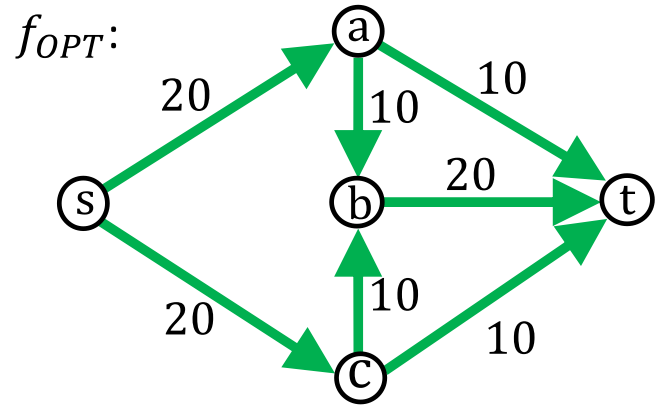
Max Flow Algorithm





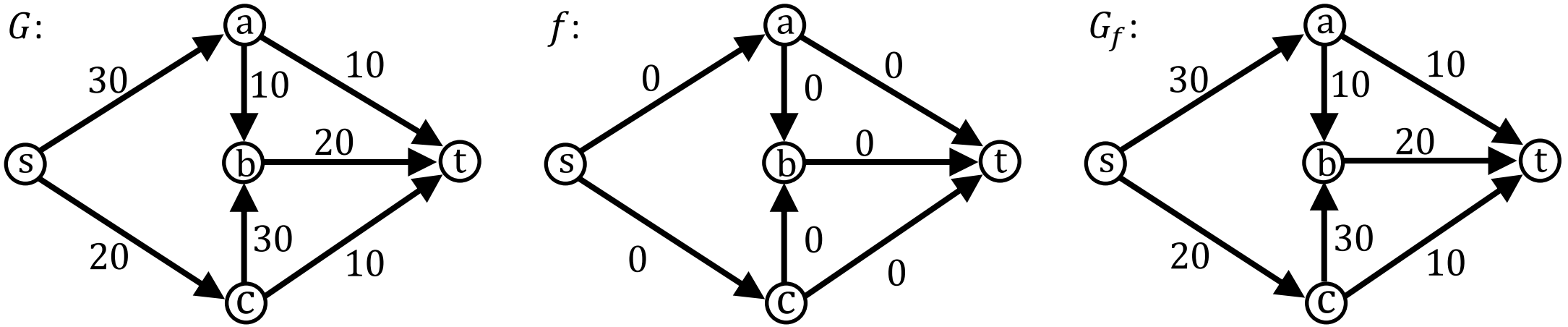


We need some way to reroute flow.



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

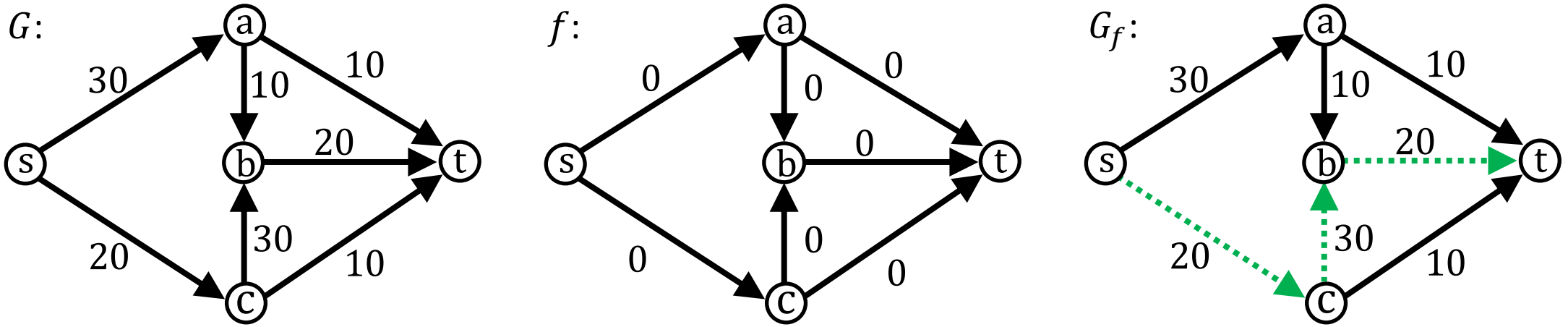
Max Flow Algorithm



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.

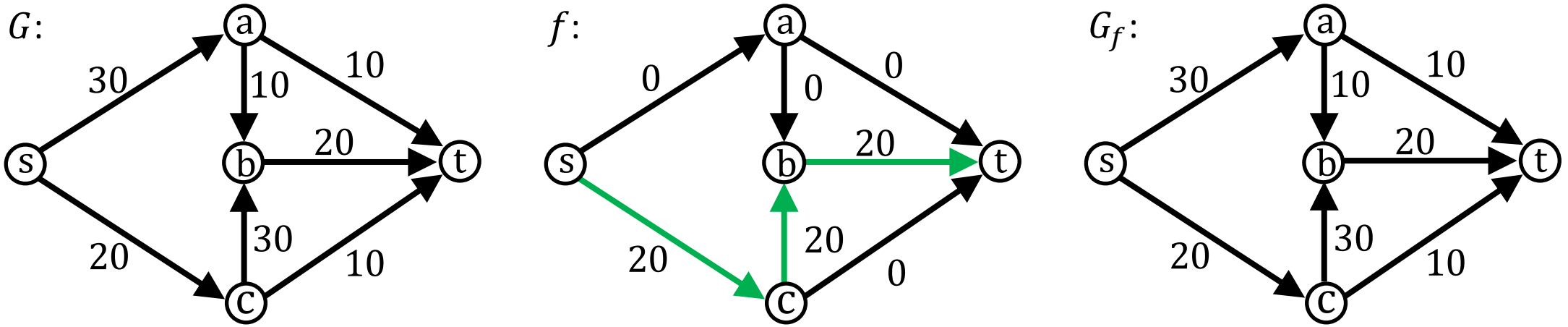
Max Flow Algorithm



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.

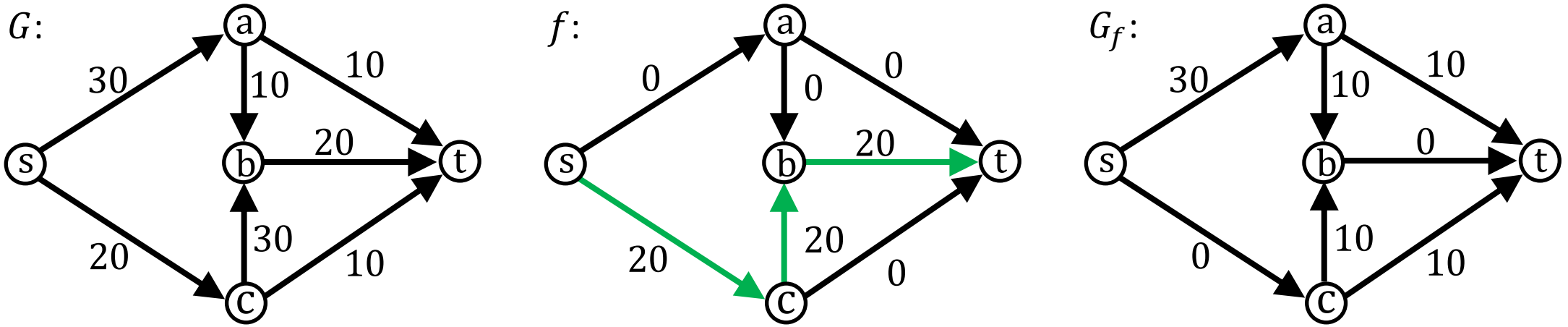
Max Flow Algorithm



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.

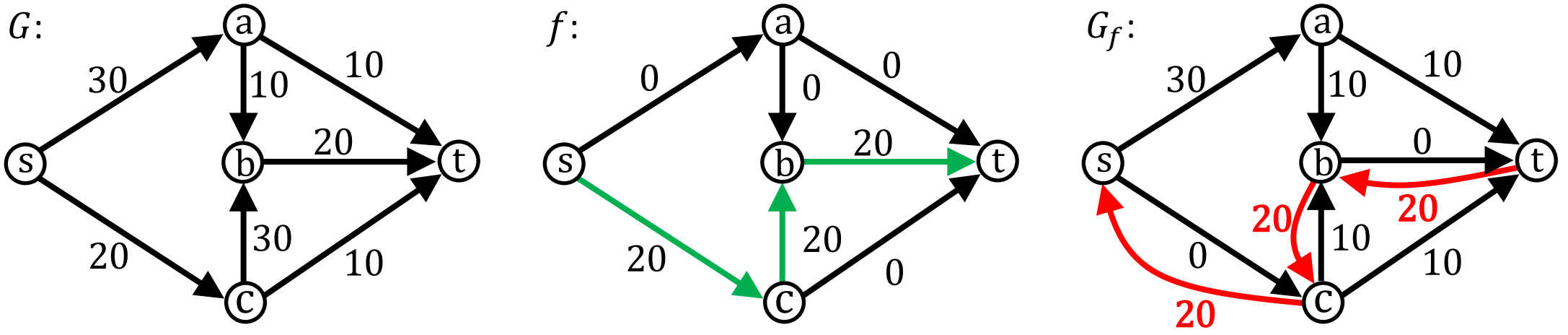
Max Flow Algorithm



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.

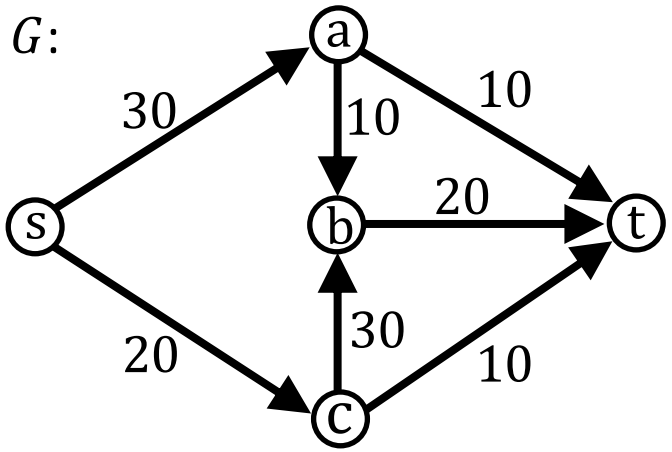
Max Flow Algorithm

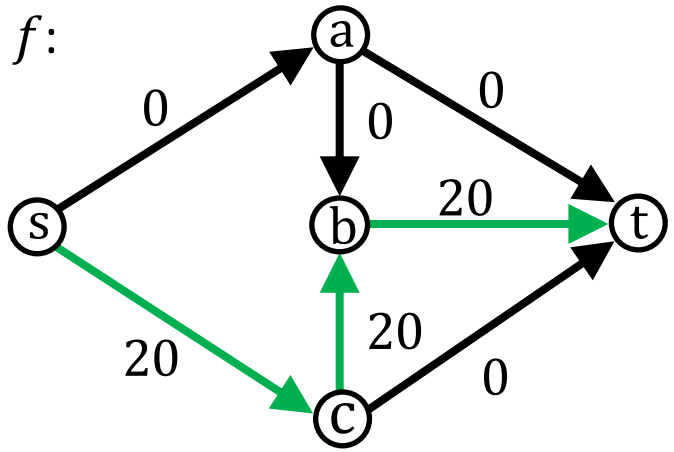


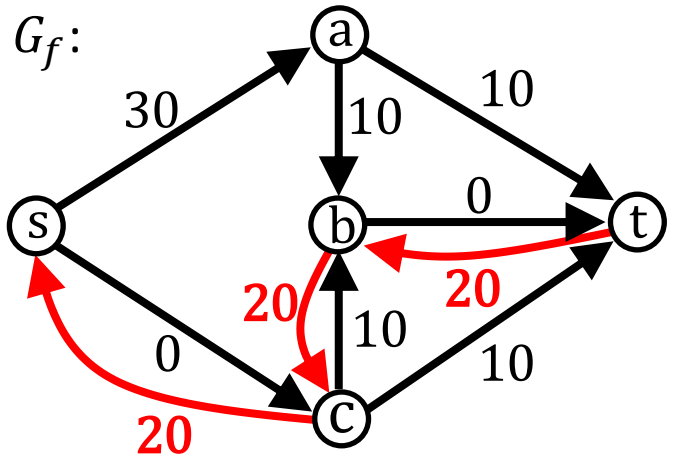
Algorithm Overview

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





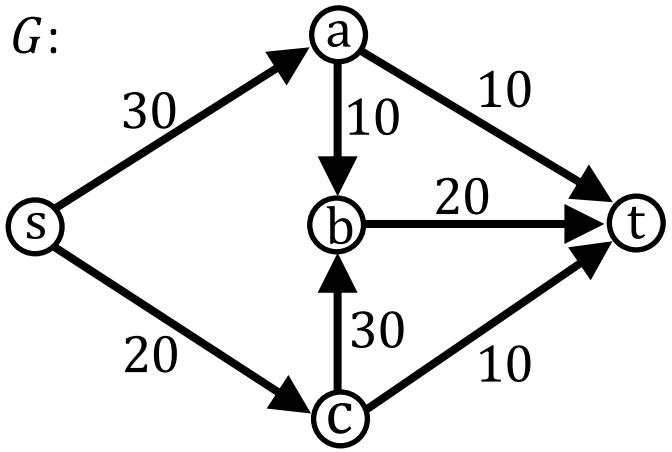


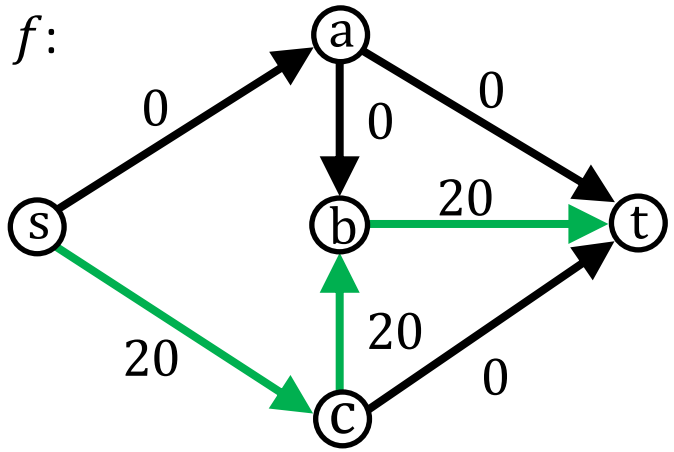
Algorithm Overview

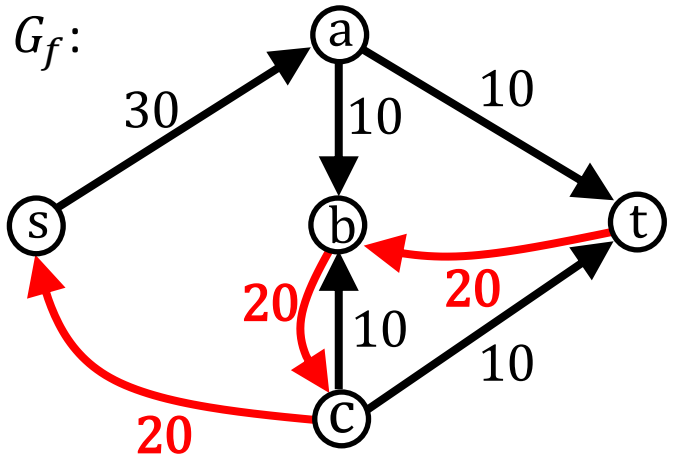
1. Start with 0 flow and initial residual graph.
2. Select an $s - t$ path P in residual graph.
3. Push bottleneck(P) flow on P.
4. Update residual graph.
5. Repeat until no $s - t$ paths exist in residual graph.

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

Max Flow Algorithm





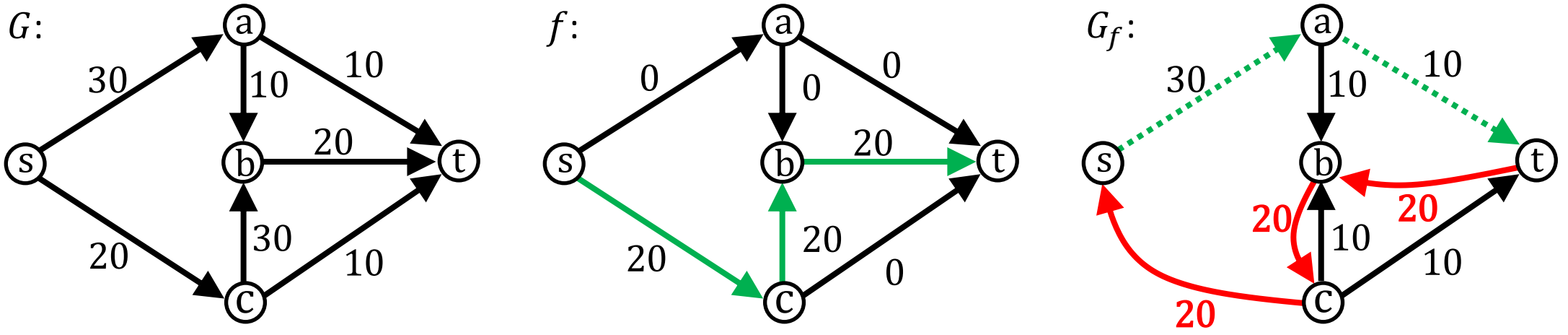


Algorithm Overview

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.

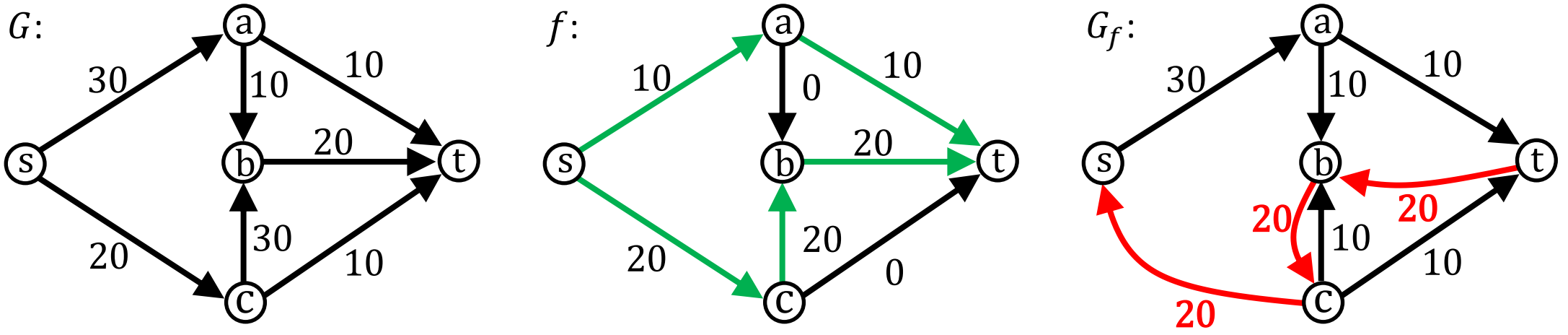
Max Flow Algorithm



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.

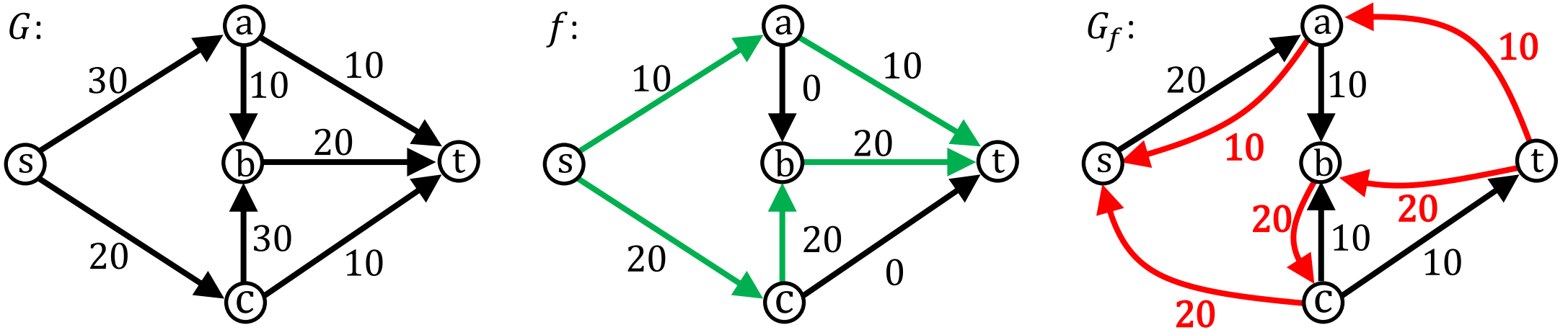
Max Flow Algorithm



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.

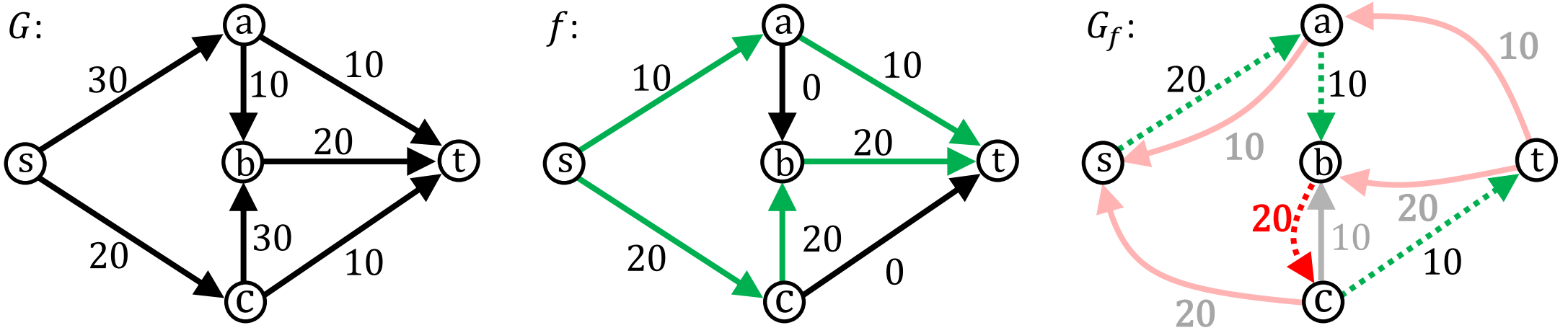
Max Flow Algorithm



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.

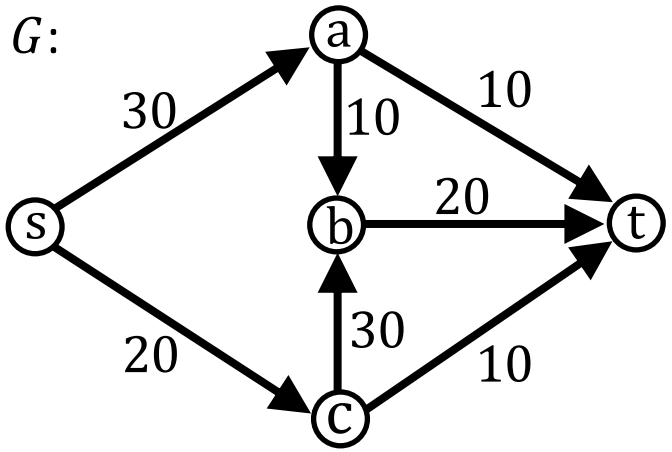
Max Flow Algorithm

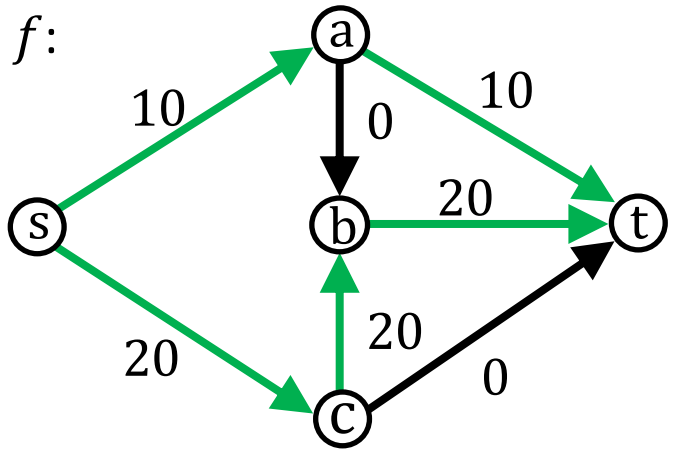


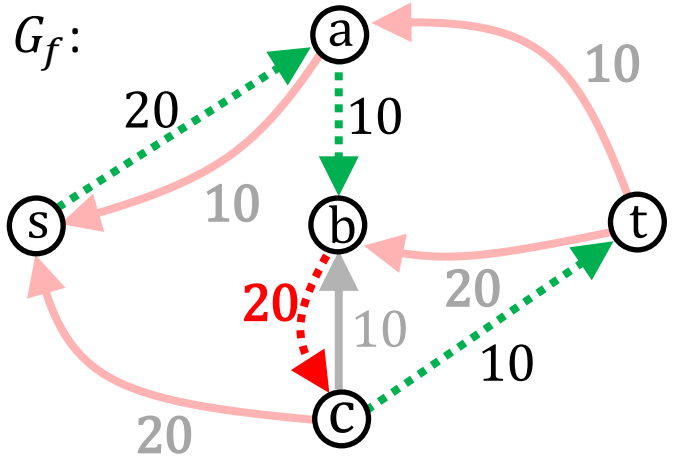
Algorithm Overview

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





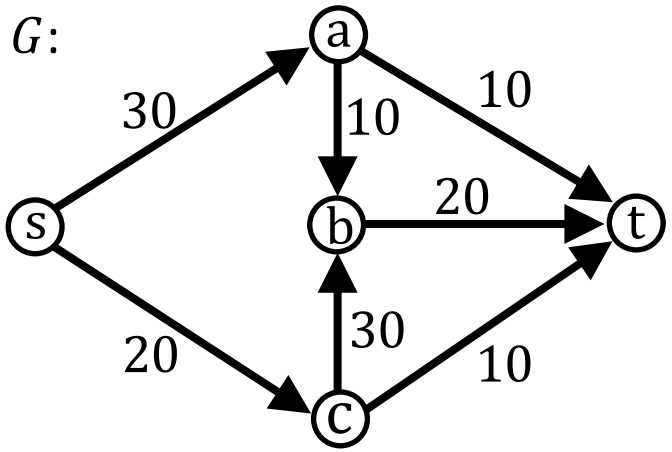


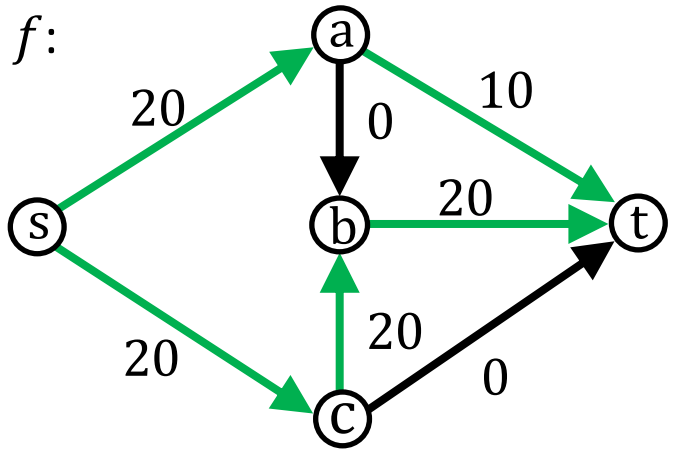
Pushing flow on a path:

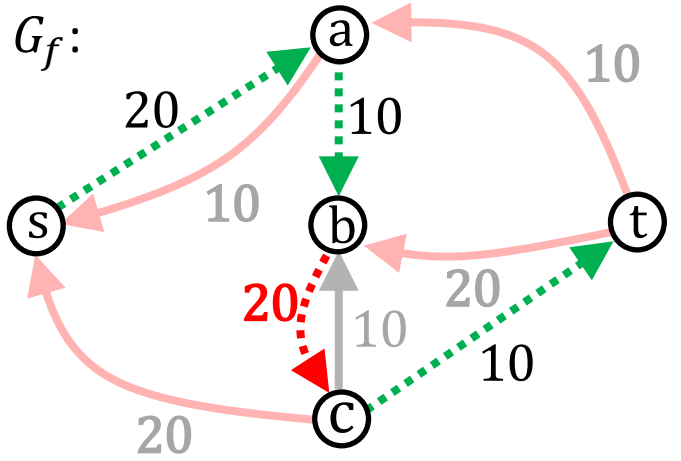
Algorithm Overview

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







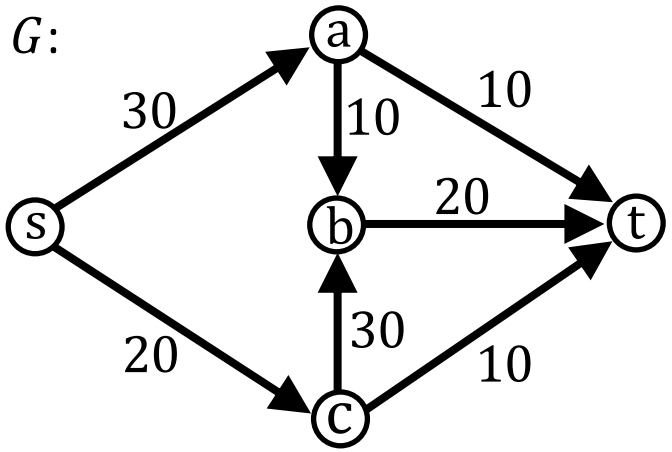
Pushing flow on a path:

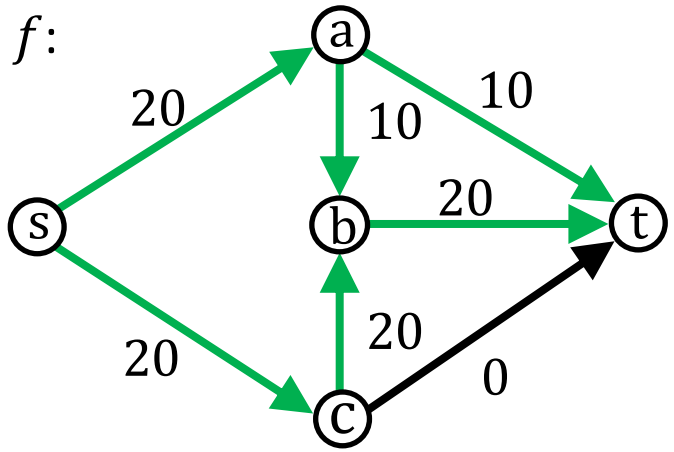
1. If edge is a normal edge, increase flow.

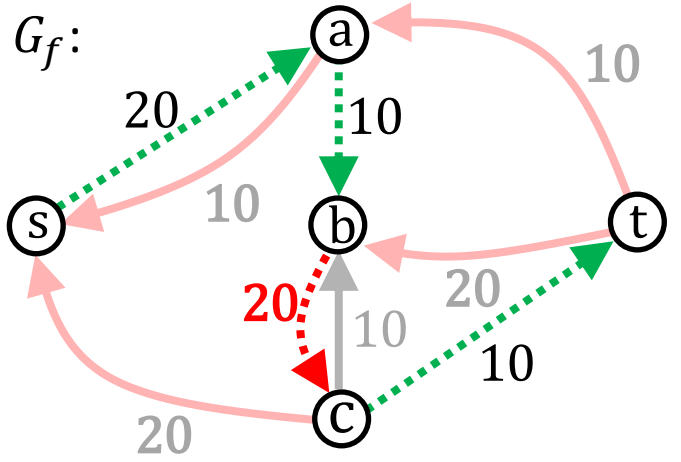
Algorithm Overview

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







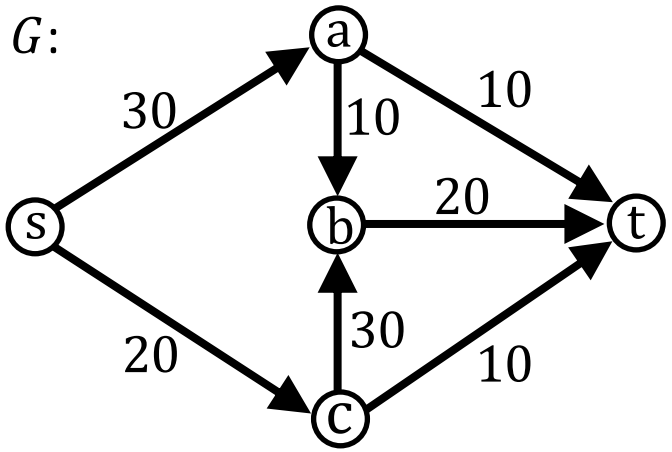
Pushing flow on a path:

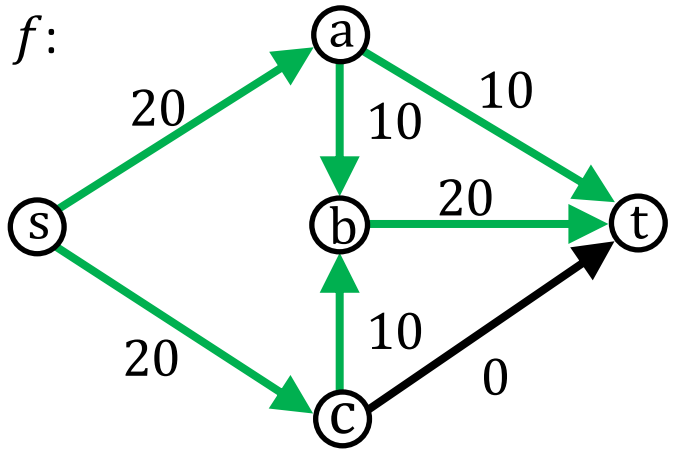
1. If edge is a normal edge, increase flow.

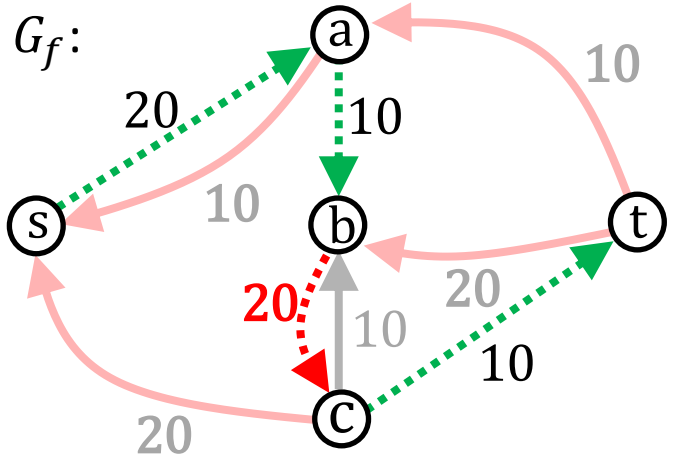
Algorithm Overview

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







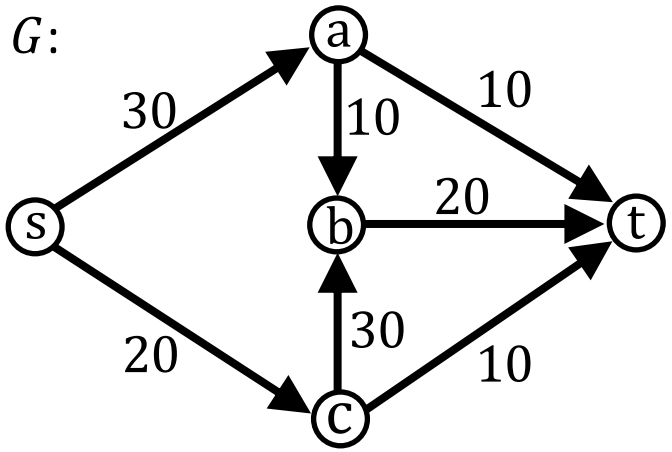
Algorithm Overview

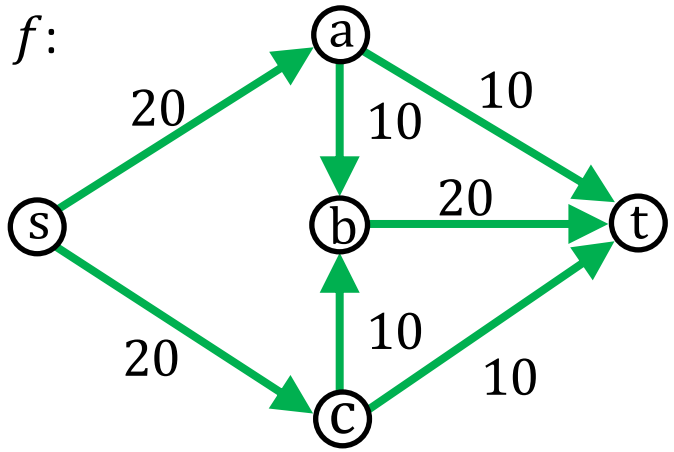
1. Start with 0 flow and initial residual graph.
2. Select an $s - t$ path P in residual graph.
3. Push bottleneck(P) flow on P.
4. Update residual graph.
5. Repeat until no $s - t$ paths exist in residual graph.

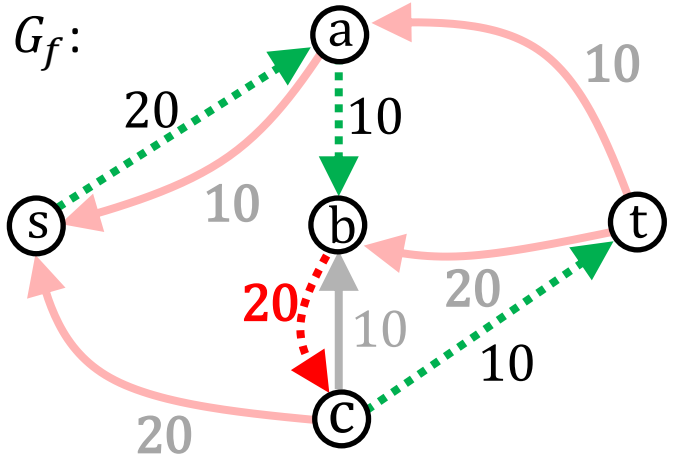
Pushing flow on a path:

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

Max Flow Algorithm







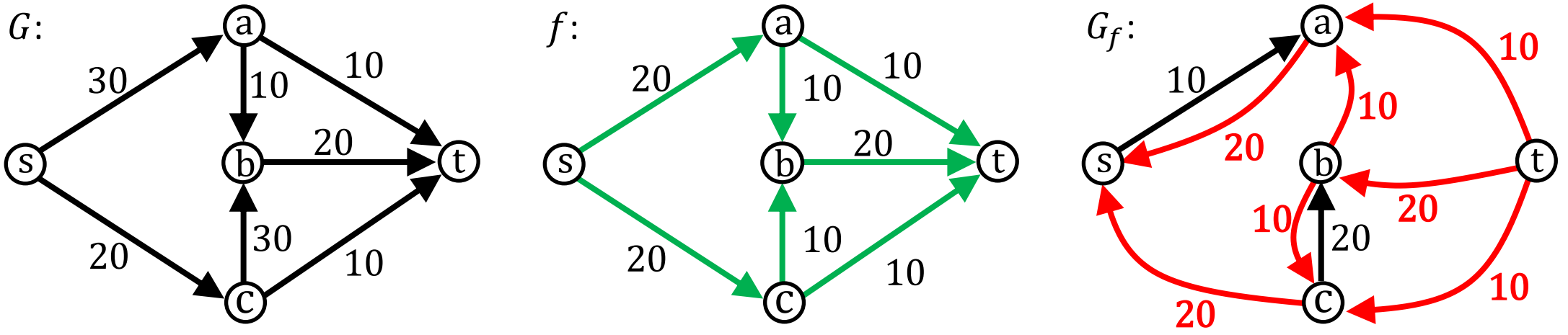
Algorithm Overview

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.

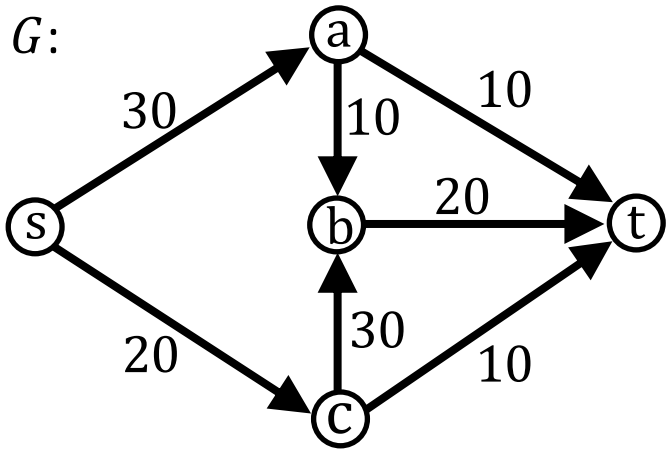
Max Flow Algorithm

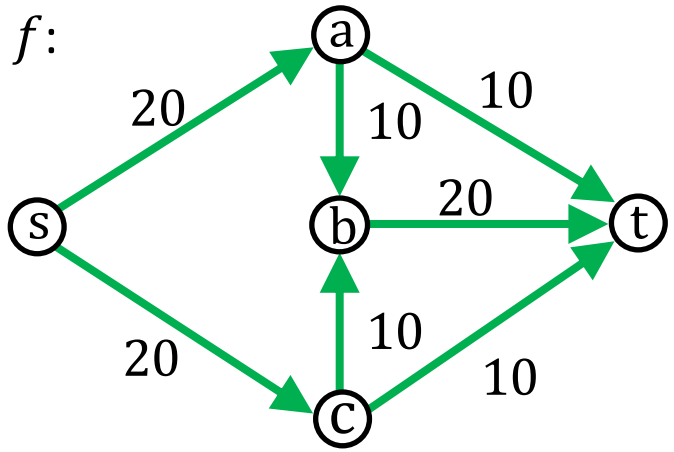


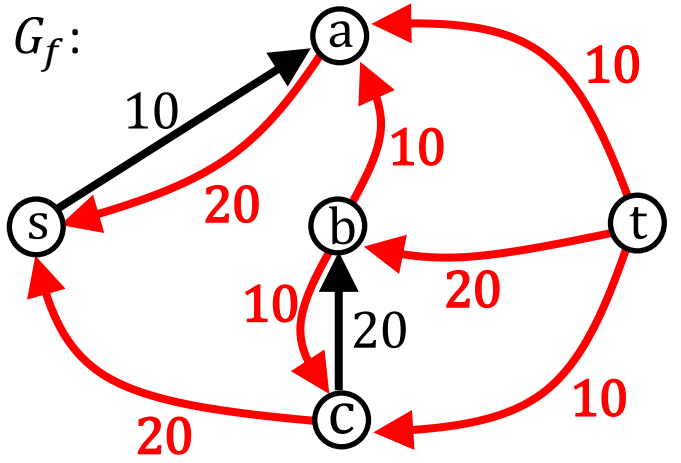
Algorithm Overview

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



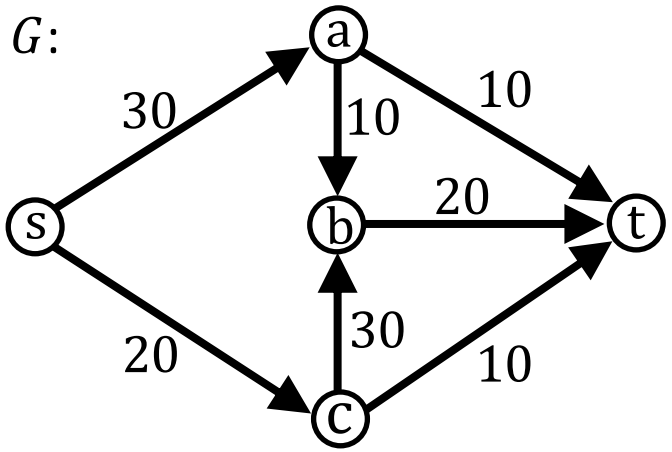


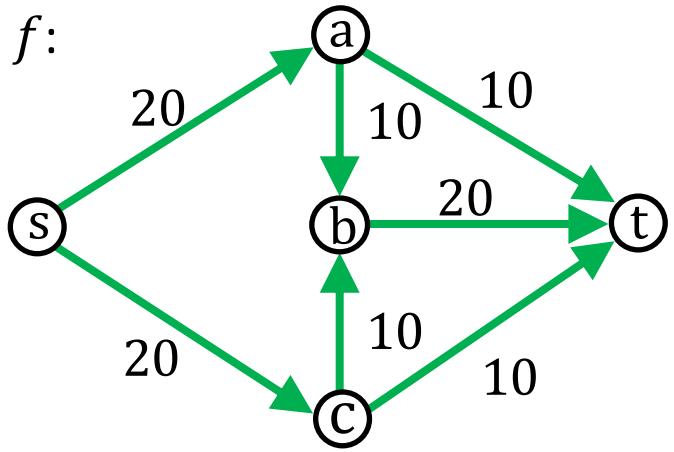


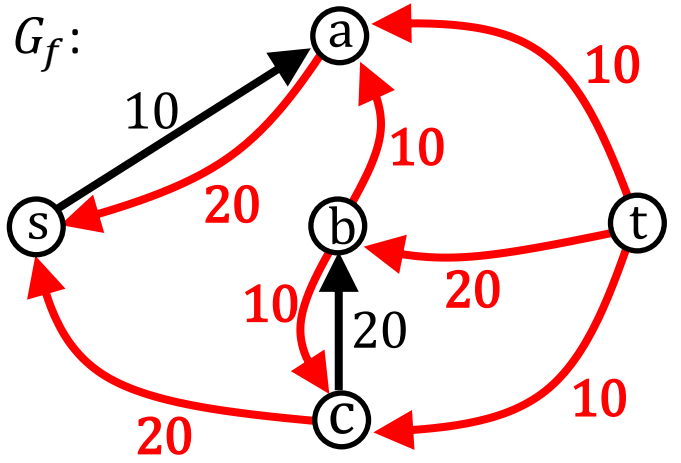
Algorithm Overview

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

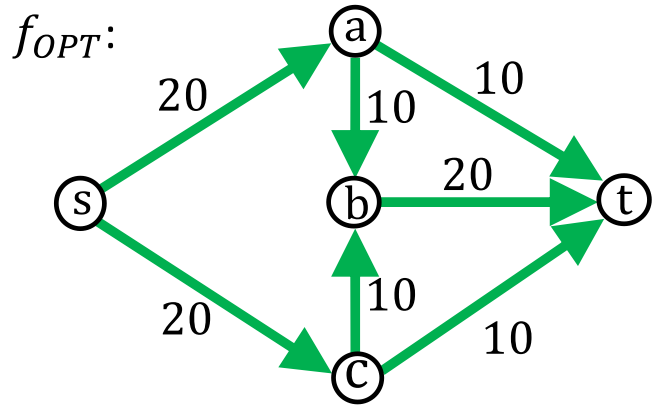




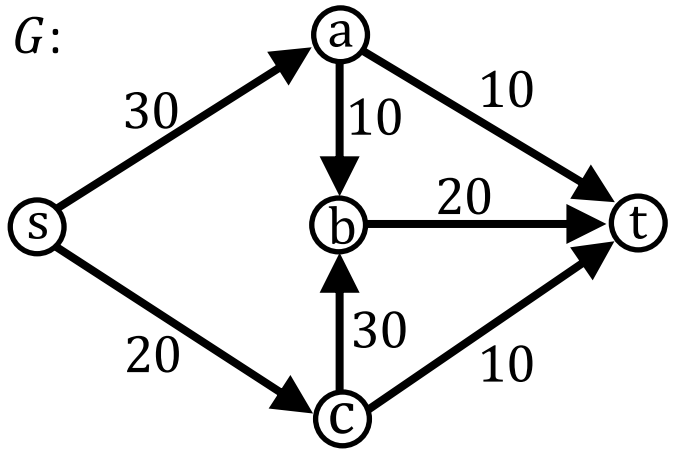


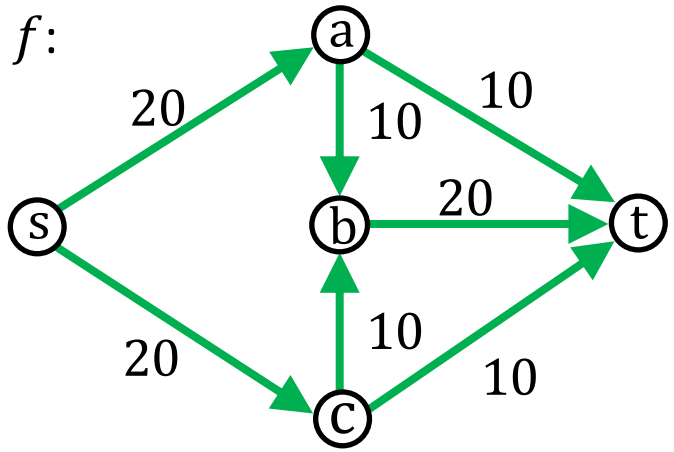
Algorithm Overview

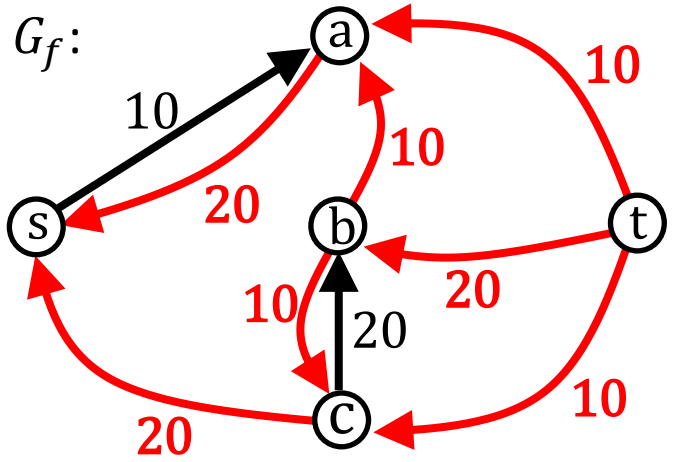
1. Start with 0 flow and initial residual graph.
2. Select an $s - t$ path P in residual graph.
3. Push bottleneck(P) flow on P.
4. Update residual graph.
5. Repeat until no $s - t$ paths exist in residual graph.



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 Behavior

Claims:

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

?

```
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 Behavior

Claims:

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

$$\text{Flow} = 0$$
$$\&$$
$$\text{Capacities} \in \mathbb{N}$$

```
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
```

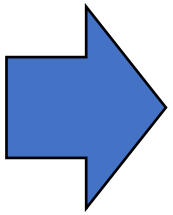
```
augment(f, P)
   b = bottleneck(P,f)
   for each edge (u, v) in P
      if (u, v) is a back edge
         f((v, u)) -= b
      else
         f((u, v)) += b
   return f
```

Ford-Fulkerson Behavior

Claims:

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

Flow $= 0$
&
Capacities $\in \mathbb{N}$



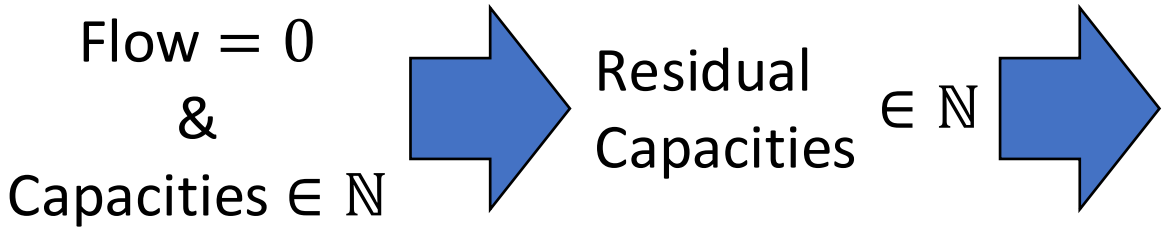
```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = simple s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

```
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 Behavior

Claims:

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



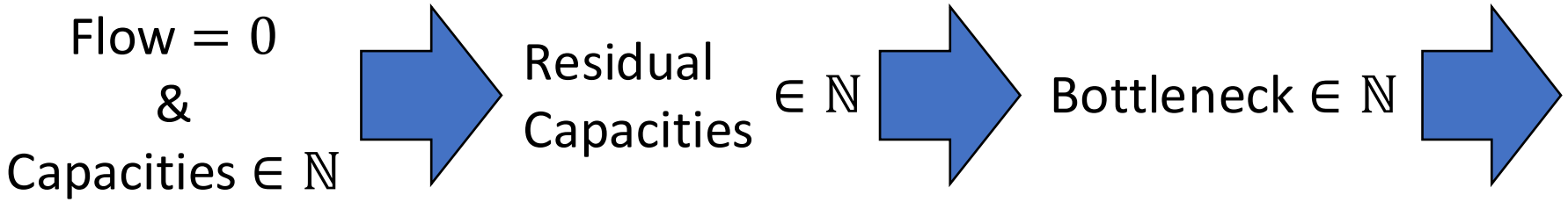

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = simple s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

```
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 Behavior

Claims:

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



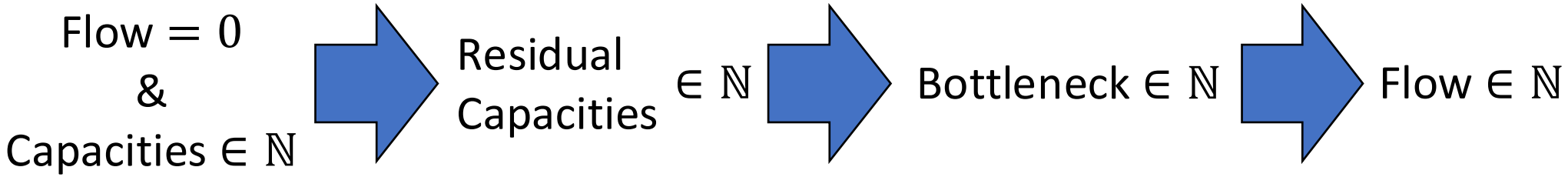

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = simple s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

```
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 Behavior

Claims:

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



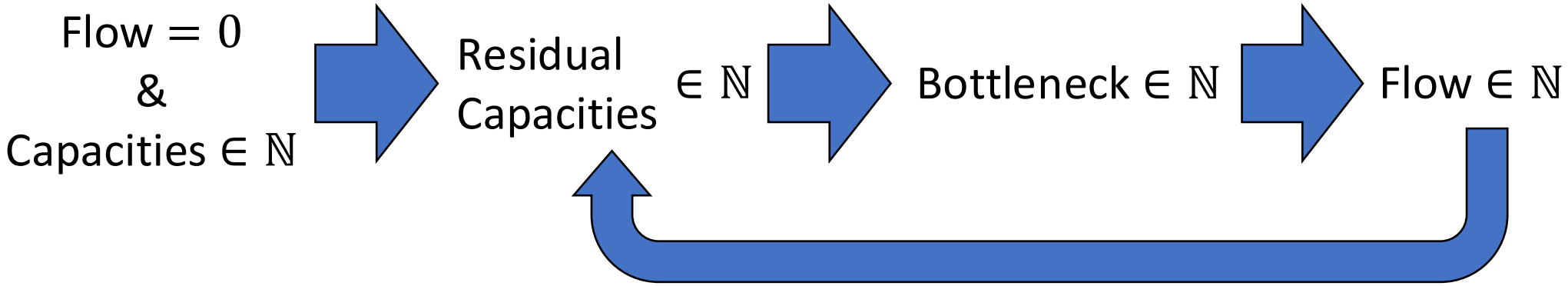

```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = simple s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

```
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 Behavior

Claims:

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




```
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 Behavior

Claims:

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

?

```
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 Behavior

Claims:

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

   With integer capacities, each iteration increases flow by ≥ 1.

```
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 Behavior

Claims:

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, the algorithm needs $\leq$ |Max Flow| iterations.

```
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 Behavior

Claims:

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, 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).

```
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 Behavior

Claims:

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

   With … ≥ 1. Since the max … iterations.

How do we find max flows?

We'll get to that.

Assumptions for now:

1. We can find max flows for valid flow networks.
2. If the capacities of a flow network are integer-valued, the max flow will be too.

```
Max-F…
   f(e…
   while …
      P = sim…
      f'= augmen…
      f = f'
      Gf = Gf'
   return f
```

```
… in P
… a back edge
…) -= b
else
   f((u, v)) += b
return f
```

Ford-Fulkerson Behavior

Claims:

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

   With ... ≥ 1. Since the max ... iterations.

~~How do we find max flows?~~

~~We'll get to that.~~

Ford-Fulkerson

...ios).

Assumptions for now:

1. We can find max flows for valid flow networks.
2. If the capacities of a flow network are integer-valued, the max flow will be too.

```
Max-F...
   f(e...
   while ...
      P = sim...                    ...) in P
      f'= augmen...                 ...s a back edge
      f = f'                        ...)) -= b
      G_f = G_f'                    else
   return f                            f((u, v)) += b
                                  return f
```

Ford-Fulkerson Behavior

Claims:

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

   With ... ≥ 1. Since the max ... iterations.

~~How do we find max flows?~~

~~We'll get to that.~~

Ford-Fulkerson

...ios).

Assumptions for now:

1. We can find max flows for valid flow networks.
2. ~~If the capacities of a flow network~~ are integer-valued, the max flow will be too.

```
Max-F...
   f(e...
   while ...                                  ) in P
      P = sim...                              ...s a back edge
      f'= augmen...                           ...)) -= b
      f = f'                                  else
      Gf = Gf'                                   f((u, v)) += b
   return f                                   return f
```

Ford-Fulkerson Behavior

Items of Interest:

- Running time.
- Validity.
- Optimality.

Claims:

2. If edge capaciti… inte…ed, th… …
   With … … the
   max… … iterations.

~~How do we find max flows?~~

~~We'll get to that.~~

Ford-Fulkerson

…ios).

Assumptions for now:

1. We can find max flows for valid flow networks.
2. ~~If the capacities of a flow network~~ are integer-valued, the max flow will be too.

```
Max-F…
   f(e…
   while …                                  ) in P
      P = sim…                              …s a back edge
      f'= augmen…                           …)) -= b
      f = f'                                else
      Gf = Gf'                                 f((u, v)) += b
   return f                                 return f
```