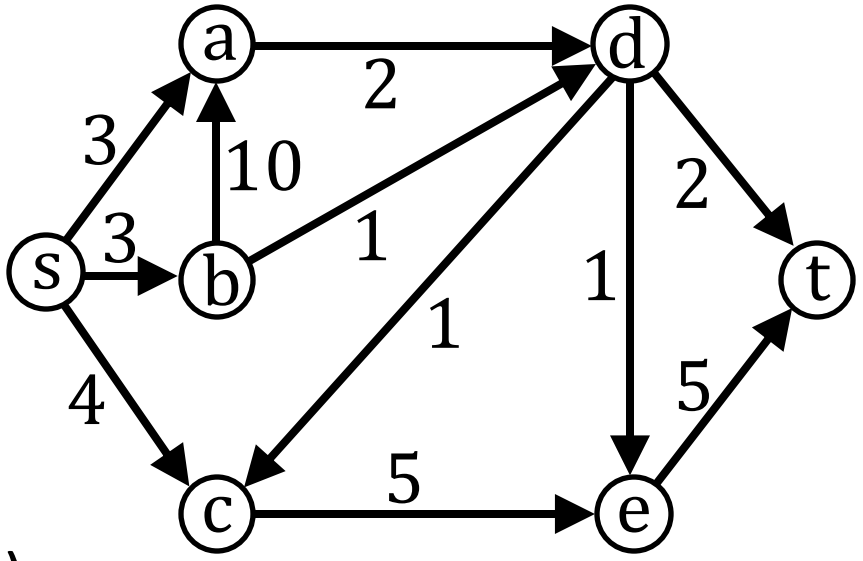
Flow Networks
CSCI 432

Flow Network

Flow Network:

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



An $s - t$ flow is a function $f\colon E \to \mathbb{R}^+$ 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 = $val(f)$ = $\sum_{e \epsilon \text{output}(s)} f(e)$ = $\sum_{e \epsilon \text{input}(t)} f(e)$

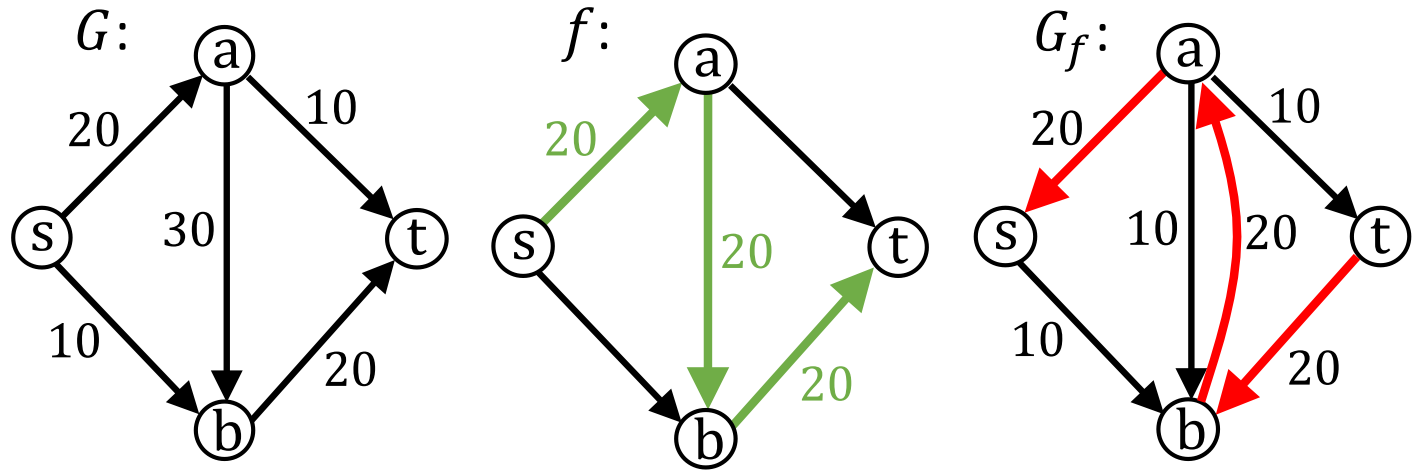
Ford-Fulkerson Algorithm

Residual graph for flow f, G_f:

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

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

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



Ford-Fulkerson Algorithm

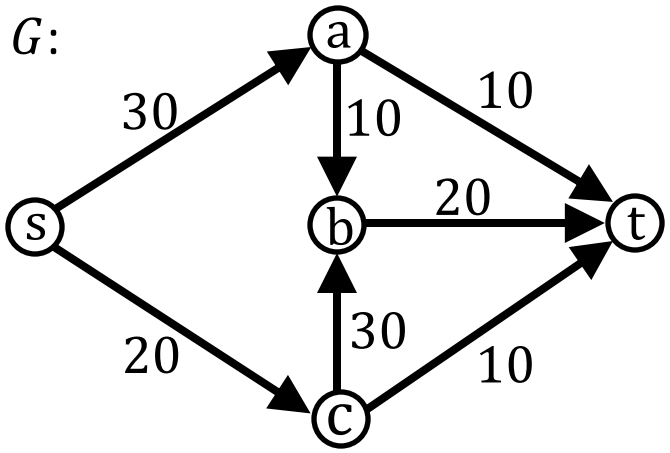
Residual graph for flow f, G_f:

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

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

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

G:



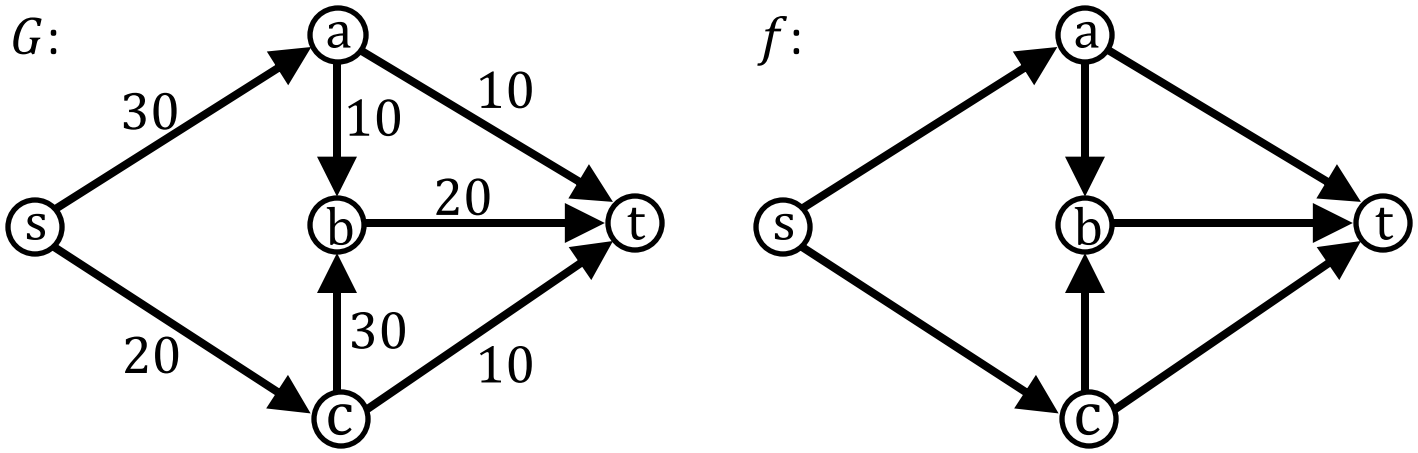
Ford-Fulkerson Algorithm

Residual graph for flow f, G_f:

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

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

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



Ford-Fulkerson Algorithm

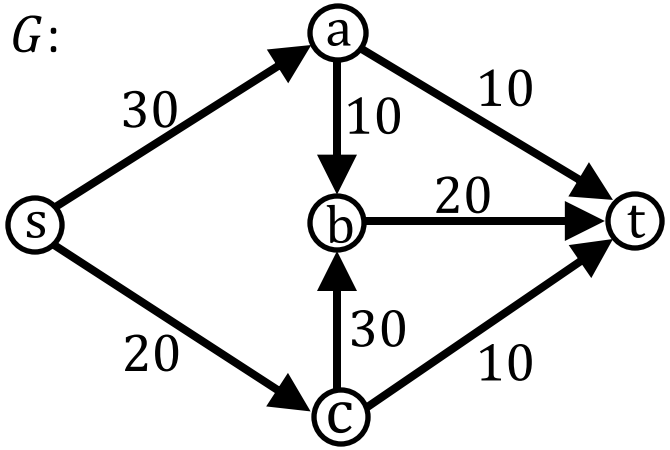
Residual graph for flow f, G_f:

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

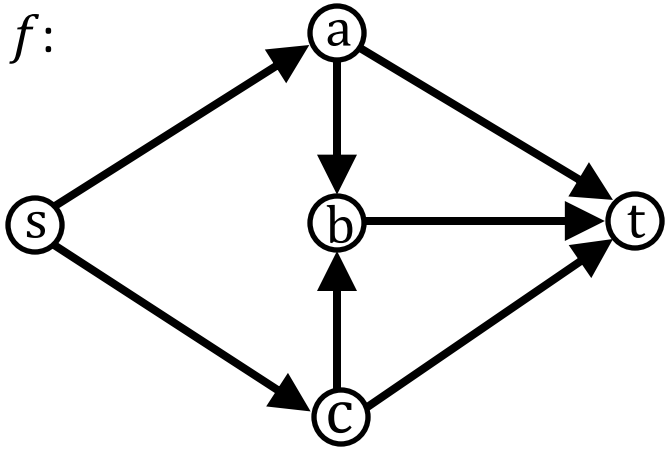
```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = simple s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

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

G:



f:



G_f:

?

Ford-Fulkerson Algorithm

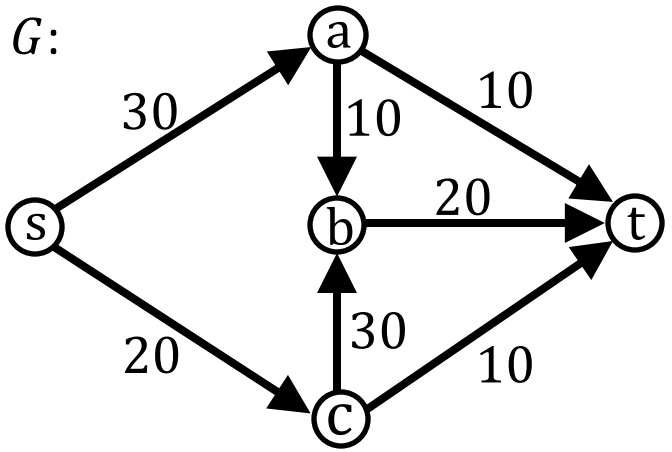
Residual graph for flow f, G_f:

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

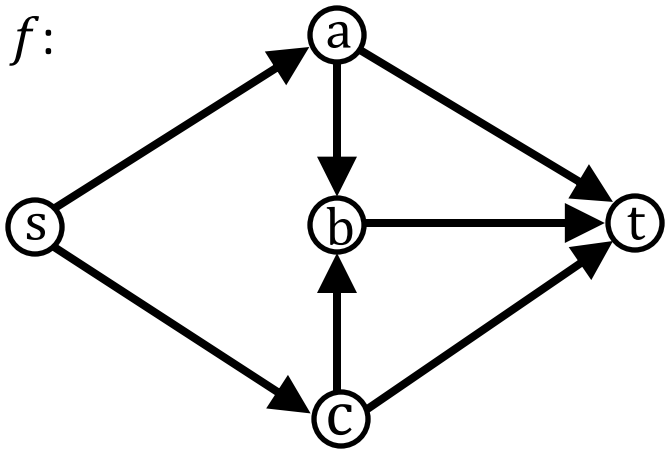
```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = simple s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

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

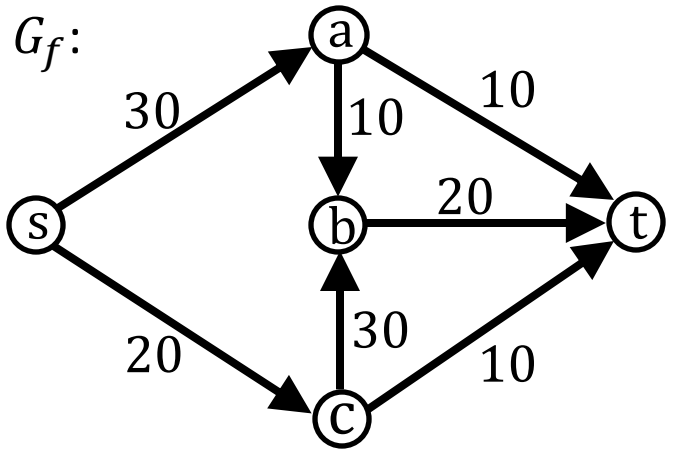
G:



f:



G_f:



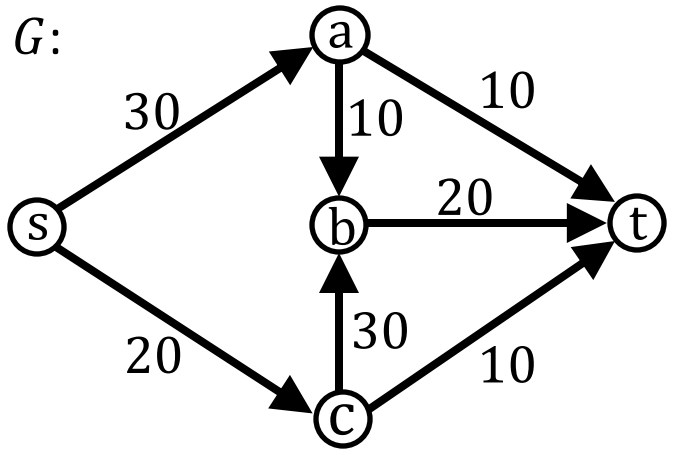
Ford-Fulkerson Algorithm

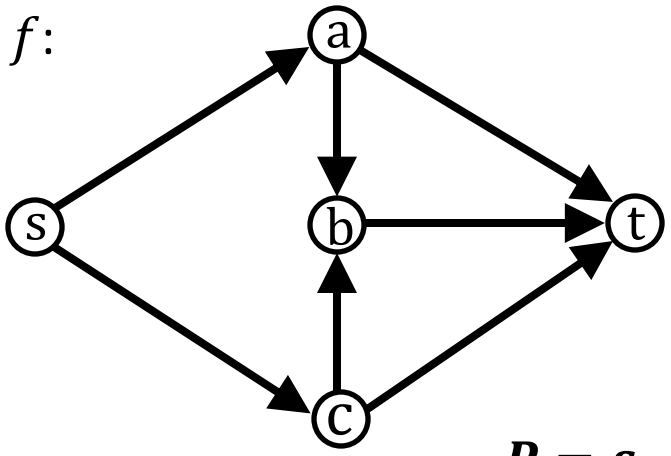
Residual graph for flow f, G_f:

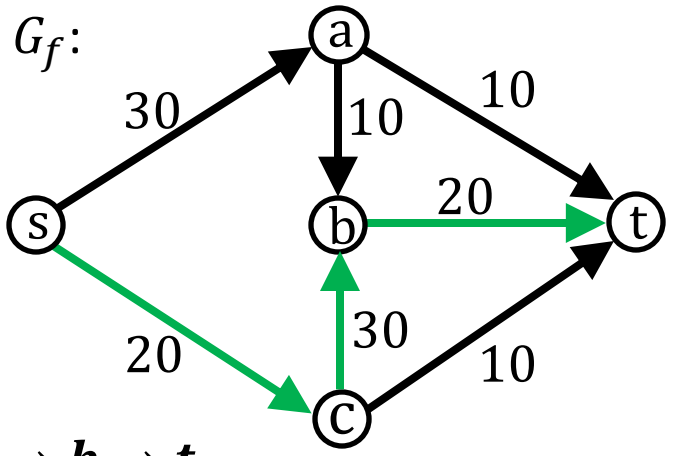
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \to c \to b \to t}$

Ford-Fulkerson Algorithm

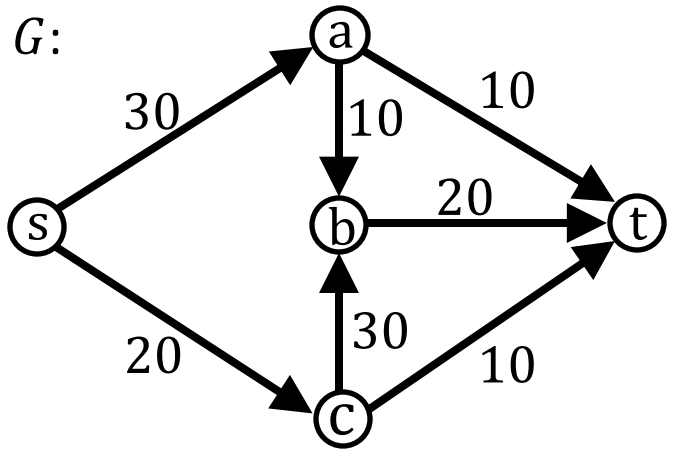
Residual graph for flow f, G_f:

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

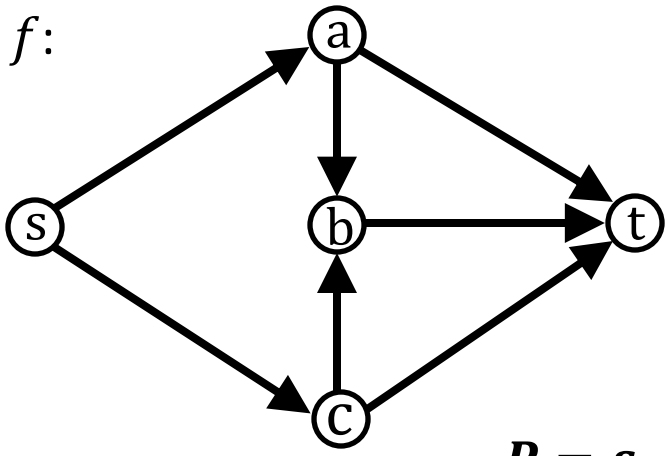
```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = simple s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

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

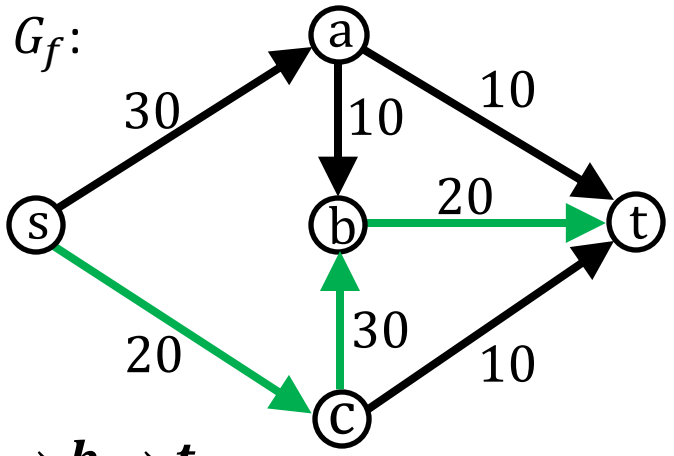
G:



f:



G_f:



$\boldsymbol{P = s \to c \to b \to t}$

Ford-Fulkerson Algorithm

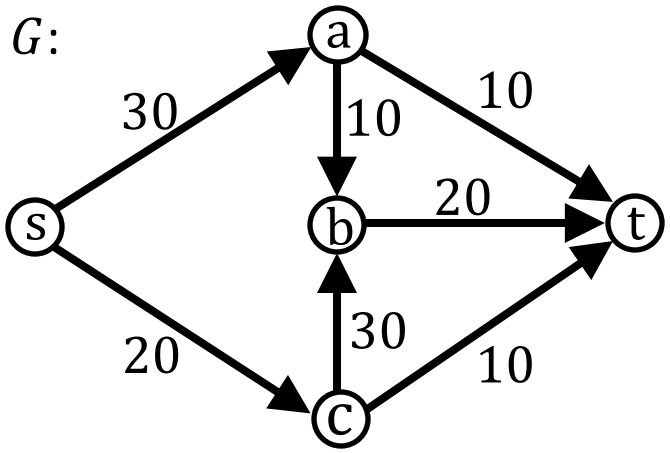
Residual graph for flow f, G_f:

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

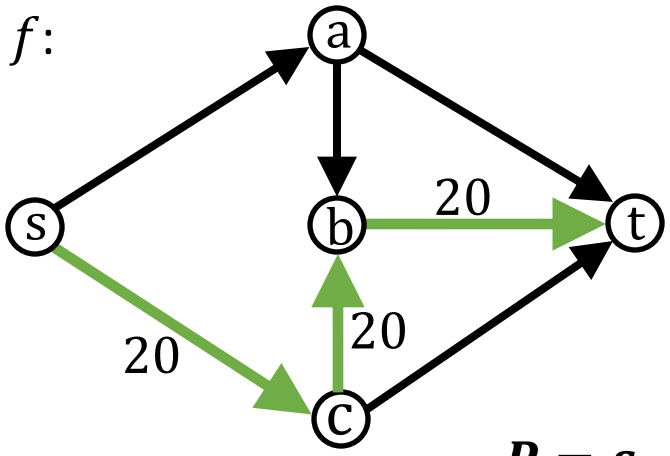
```
Max-Flow(G)
   f(e) = 0 for all e in G
   while s-t path in Gf exists
      P = simple s-t path in Gf
      f'= augment(f, P)
      f = f'
      Gf = Gf'
   return f
```

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

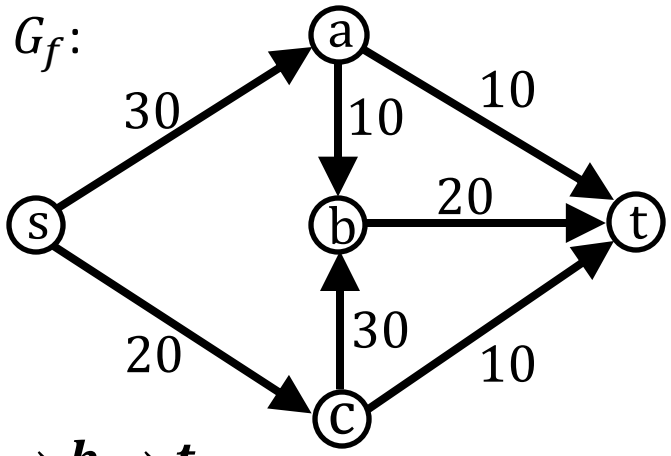
G:



f:



G_f:



$\boldsymbol{P = s \to c \to b \to t}$

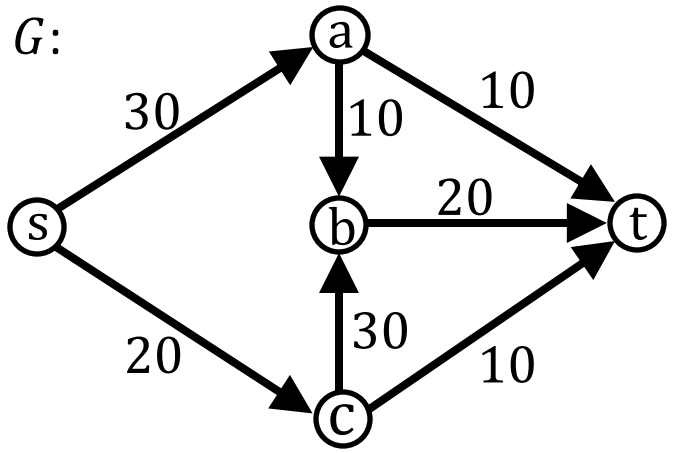
Ford-Fulkerson Algorithm

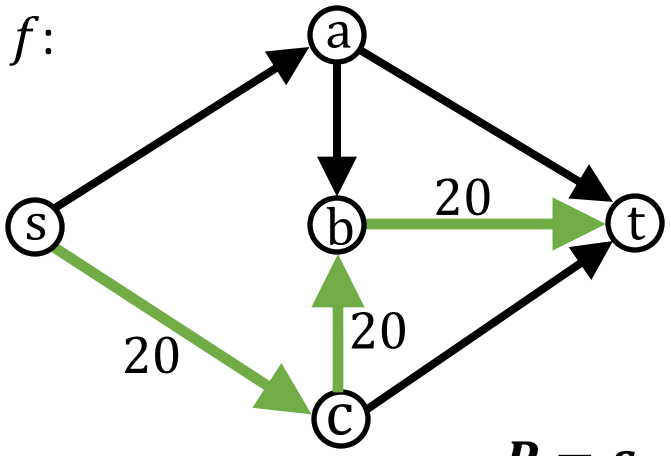
Residual graph for flow f, G_f:

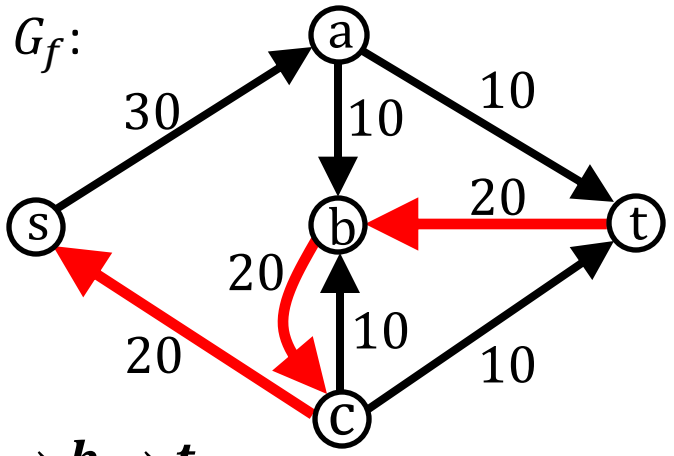
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \to c \to b \to t}$

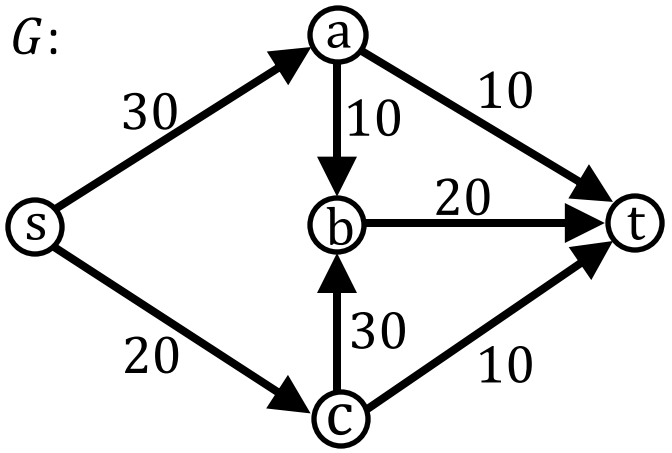
Ford-Fulkerson Algorithm

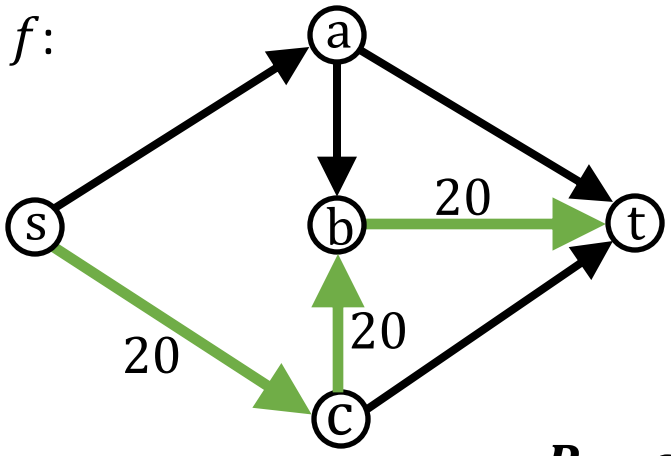
Residual graph for flow f, G_f:

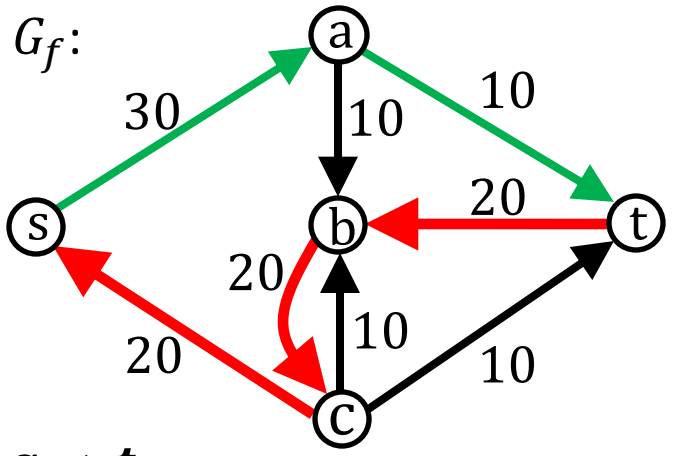
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \rightarrow a \rightarrow t}$

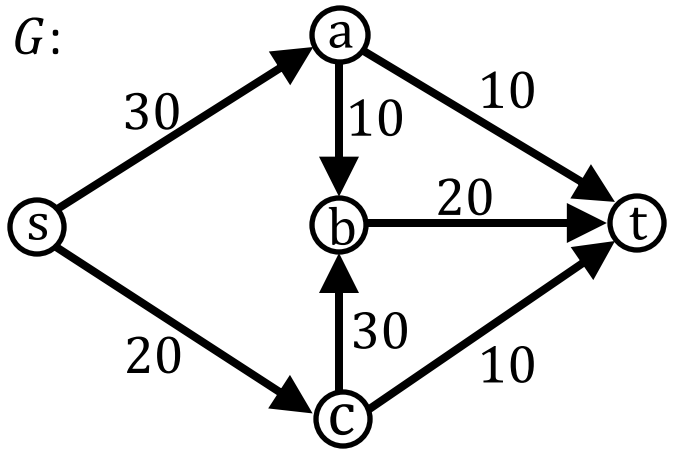
Ford-Fulkerson Algorithm

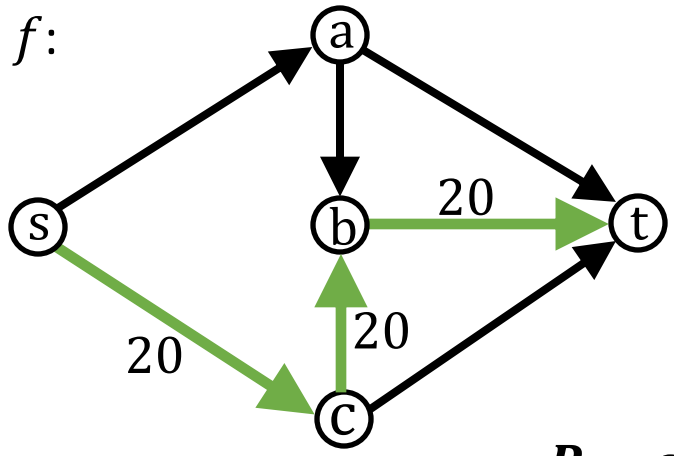
Residual graph for flow f, G_f:

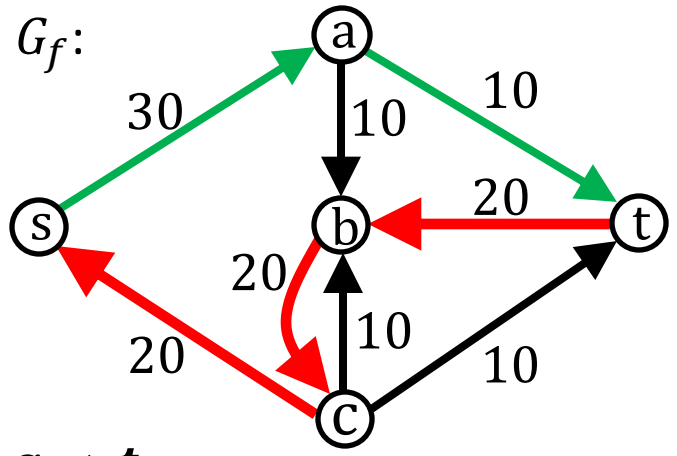
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \rightarrow a \rightarrow t}$

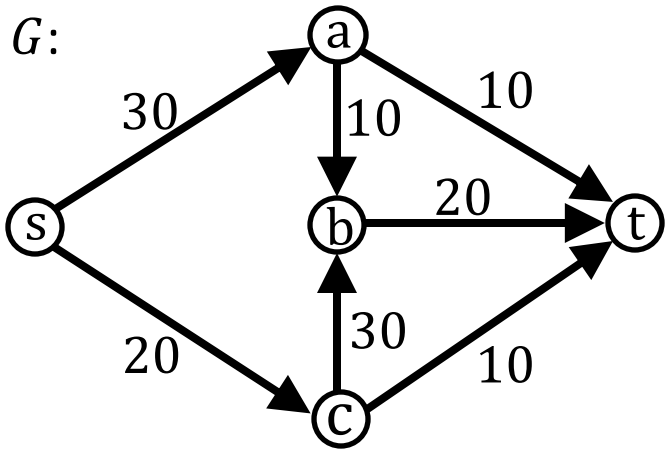
Ford-Fulkerson Algorithm

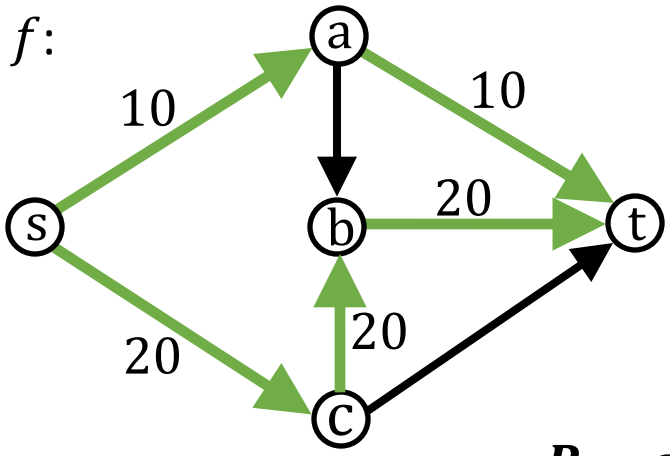
Residual graph for flow f, G_f:

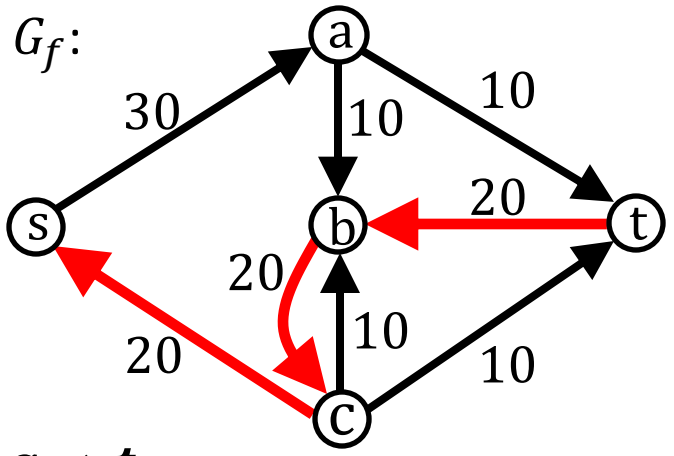
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \rightarrow a \rightarrow t}$

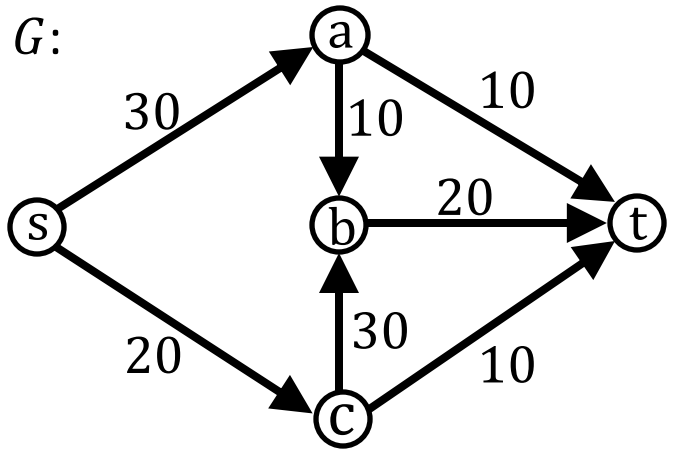
Ford-Fulkerson Algorithm

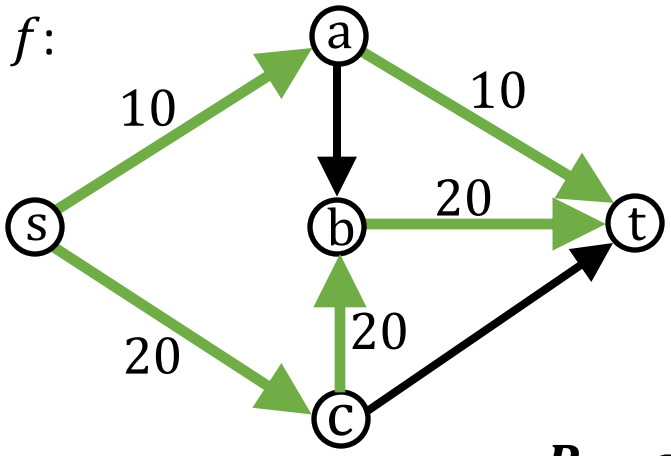
Residual graph for flow f, G_f:

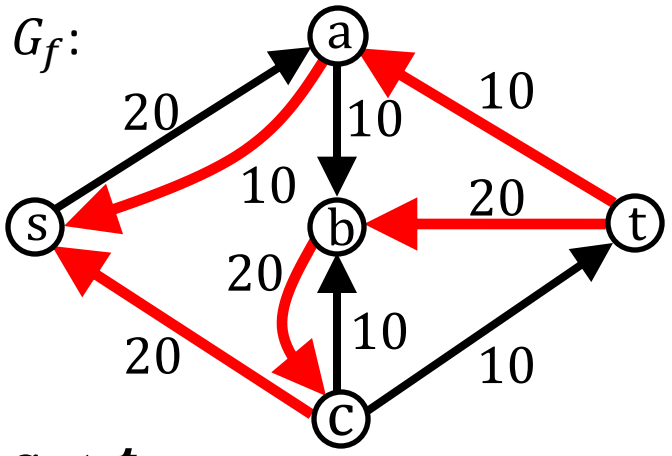
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \to a \to t}$

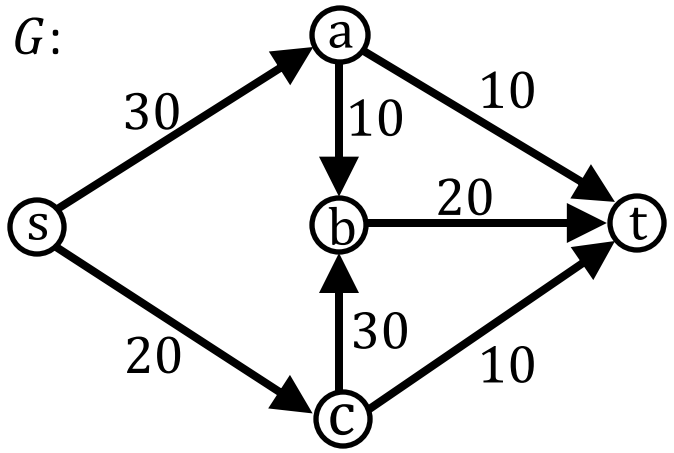
Ford-Fulkerson Algorithm

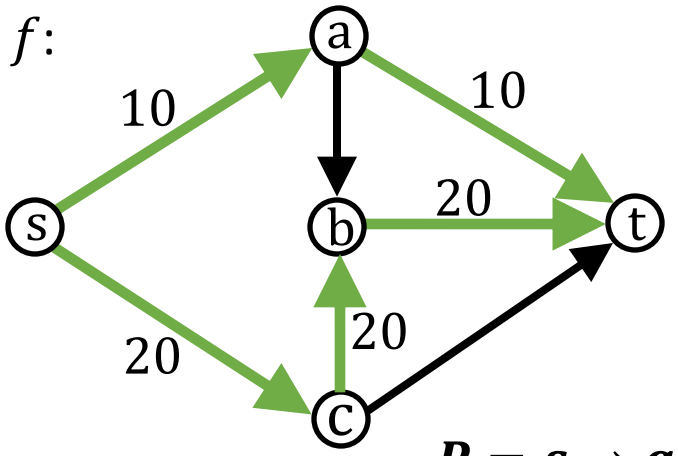
Residual graph for flow f, G_f:

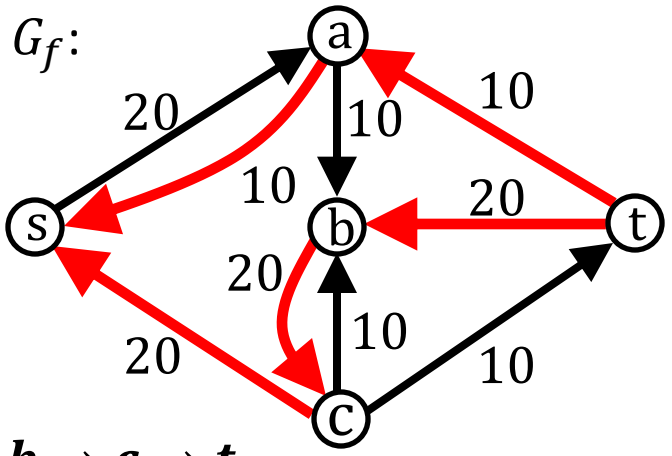
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \rightarrow a \rightarrow b \rightarrow c \rightarrow t}$

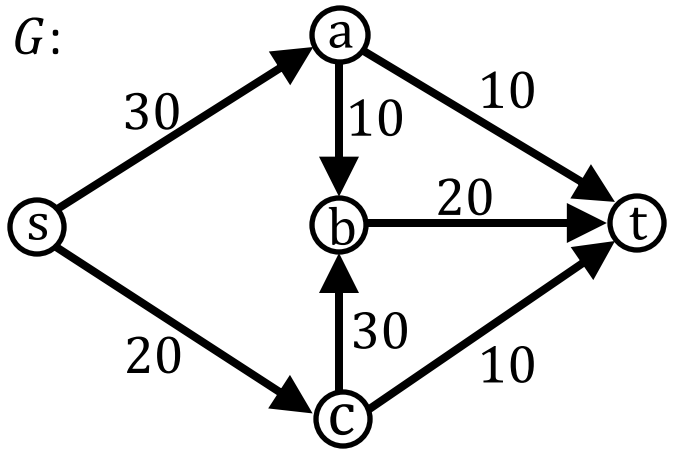
Ford-Fulkerson Algorithm

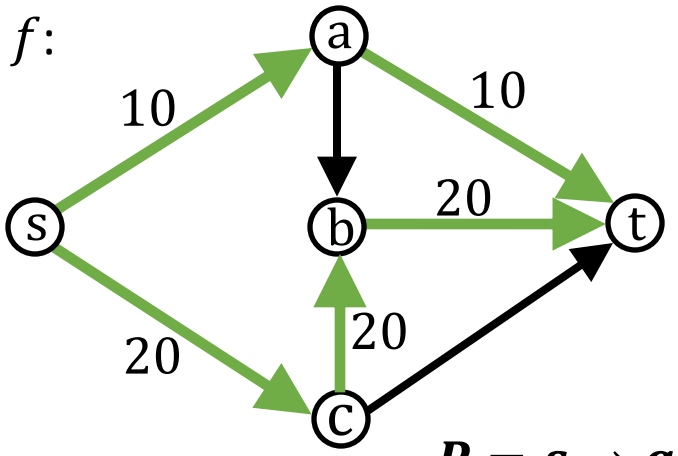
Residual graph for flow f, G_f:

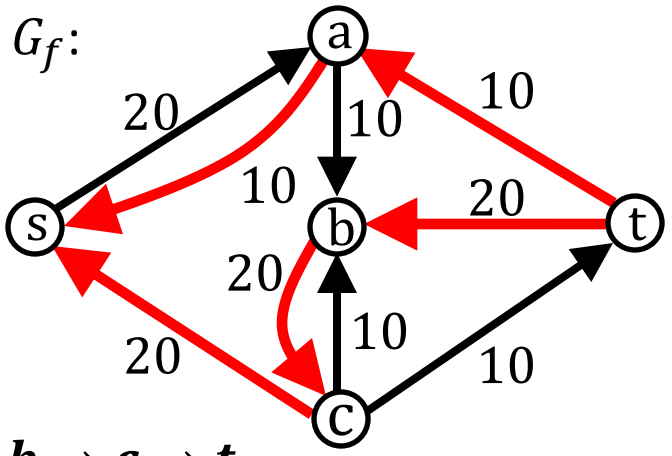
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \to a \to b \to c \to t}$

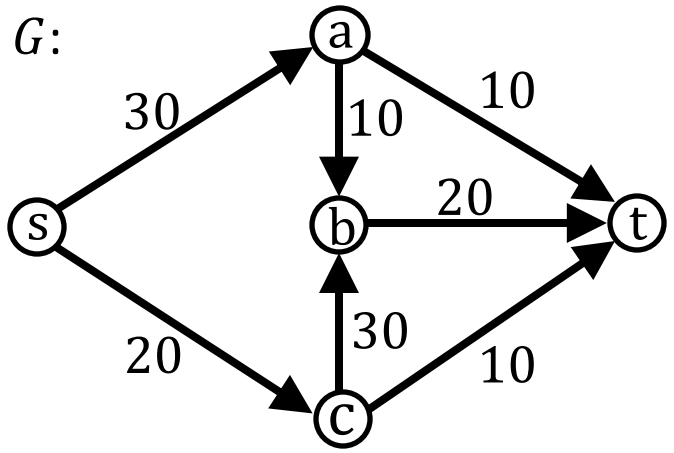
Ford-Fulkerson Algorithm

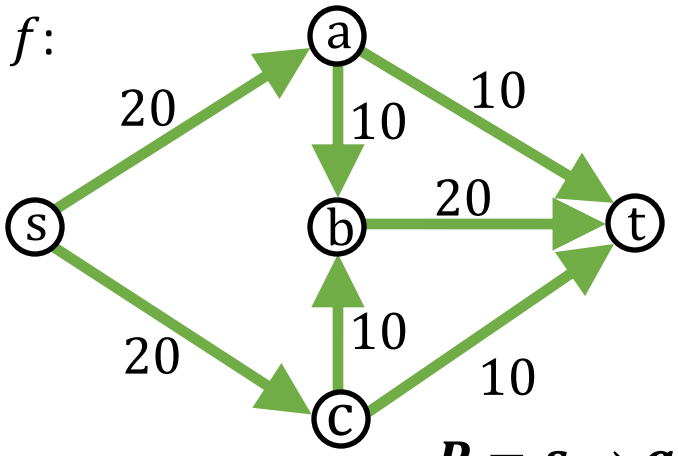
Residual graph for flow f, G_f:

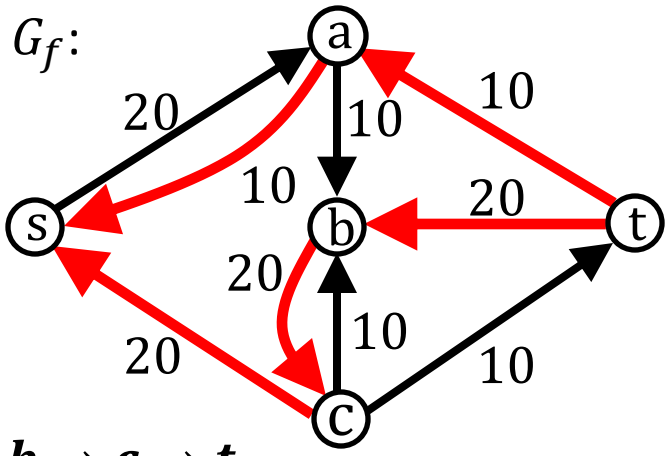
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \to a \to b \to c \to t}$

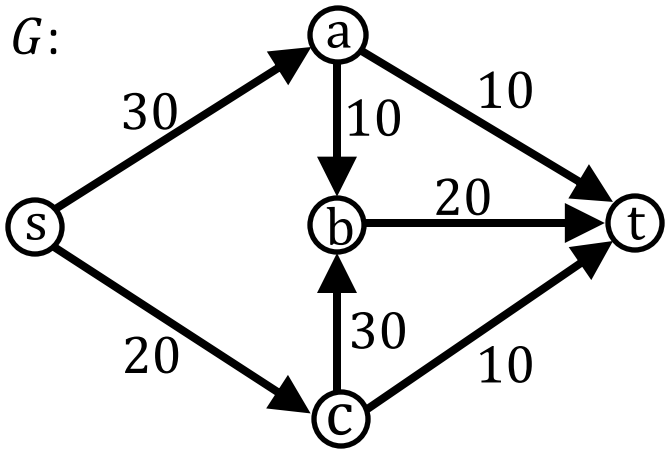
Ford-Fulkerson Algorithm

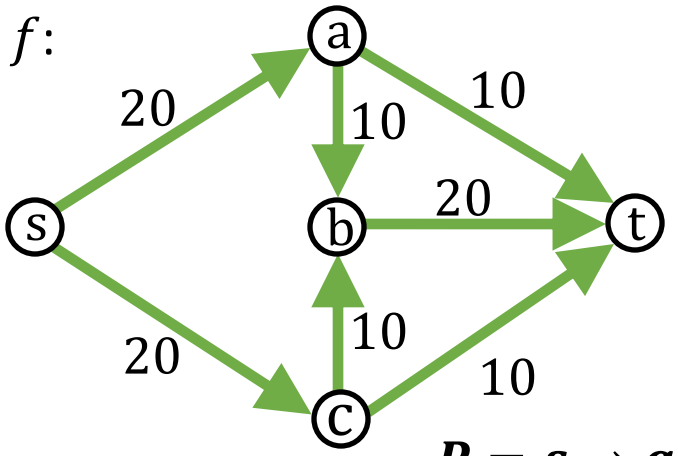
Residual graph for flow f, G_f:

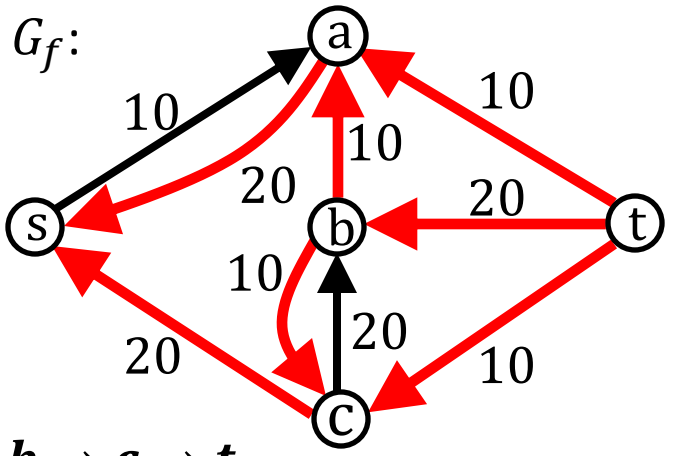
- $\forall e$, if $f(e) < c_e$, let $c_e = c_e - f(e)$.
- $\forall e = (u, v)$, if $f(e) > 0$, create $e' = (v, u)$ with $c_{e'} = f(e)$

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

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







$\boldsymbol{P = s \to a \to b \to c \to t}$

Ford-Fulkerson Algorithm

Residual graph for flow f, G_f:

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

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

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

Ford-Fulkerson Algorithm

Residual graph for flow f, G_f:

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

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

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

Need to show:

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