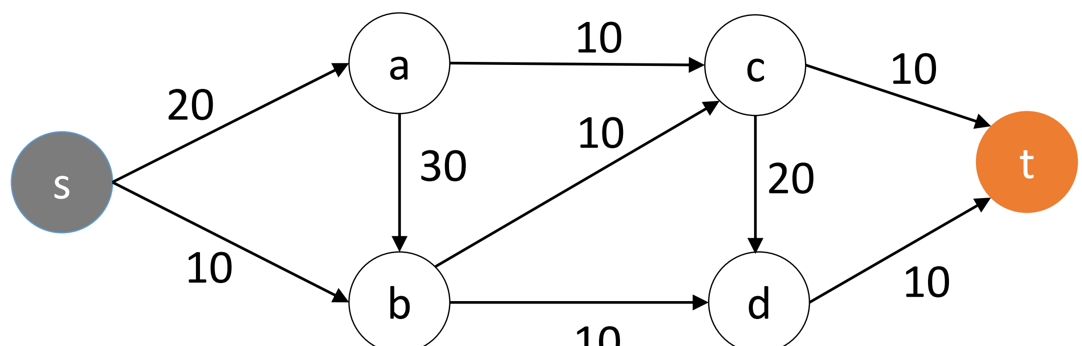
COMP261
Algorithms and Data Structures
2024 Tri 1

Jyoti Sahni
jyoti.sahni@ecs.vuw.ac.nz
Office Hours (COMP261): AM414, Thursday 10:00 – 12:00

Recap: Definition

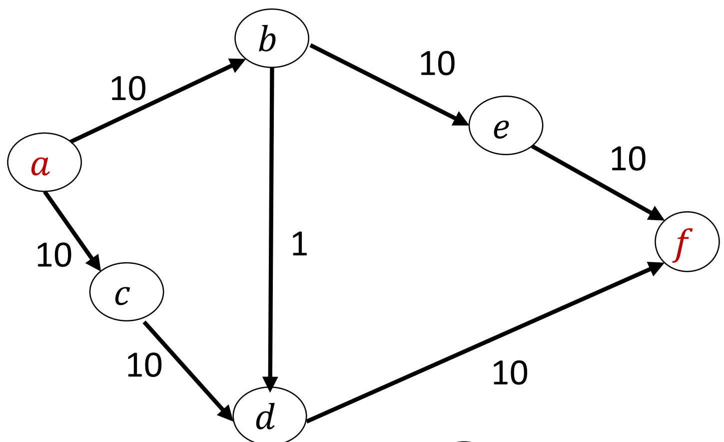
A flow network is a connected, directed graph $G = (V, E)$ where:

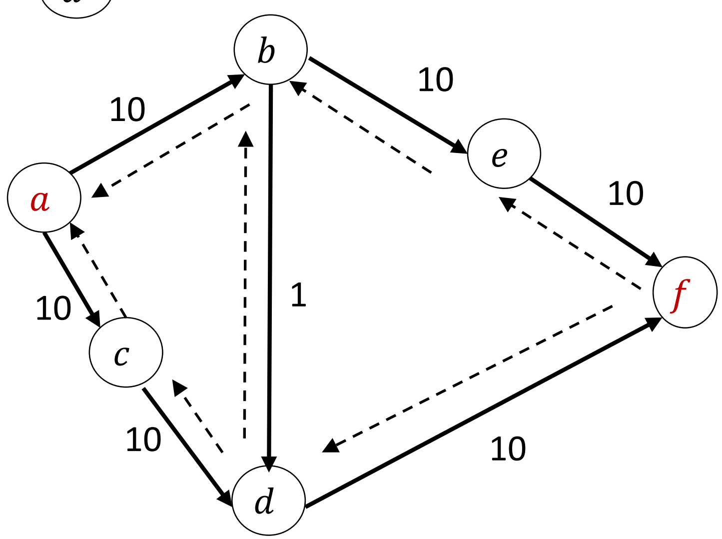
- Each edge, e, has a non-negative, integer capacity c_e.
- One (or more) vertex is labelled as a source $s \epsilon V$.
- One (or more) vertex is labelled as sink t ϵV.
- No edge enters the source and no edge leaves the sink.



Recap: Solving Maximum flow problem

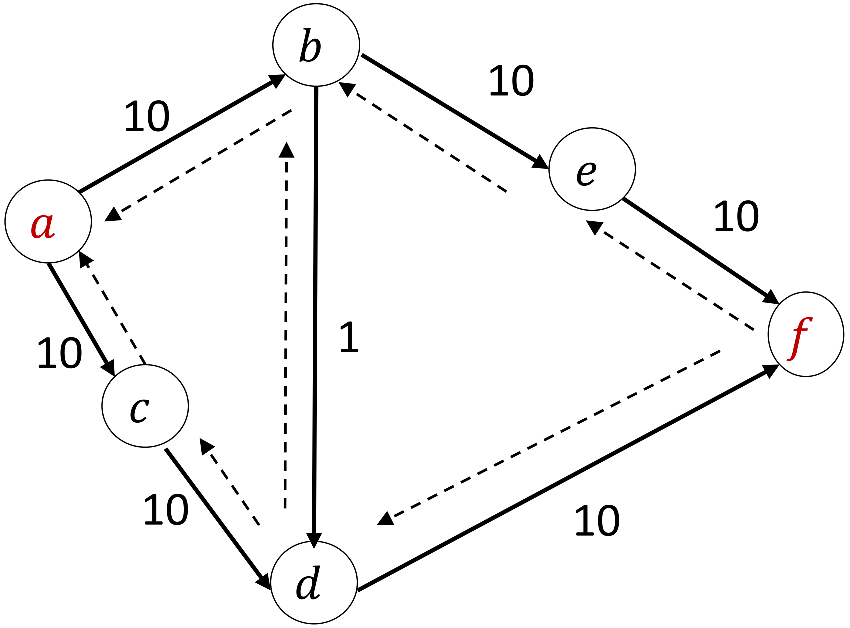
How can we solve this problem ?





1. Look for paths from source to destination and count them if they have capacities left
2. Repeat until no paths from source to destination is found with capacities left
3. There could be a possibility that the paths we choose are not optimal – have an option of reversing the decisions. For every edge in the original graph, we add an "imaginary" reverse edge

Solving Maximum flow problem



$a \dashrightarrow b \dashrightarrow d \dashrightarrow f$ 1

$a \dashrightarrow c \dashrightarrow d \dashrightarrow f$ 9

$a \dashrightarrow b \dashrightarrow e \dashrightarrow f$ 9

$a \dashrightarrow c \dashrightarrow d \dashrightarrow b \dashrightarrow e \dashrightarrow f$ 1

Ford-Fulkerson method

It is generally called a method rather than an algorithm, as it encompasses several different implementations with different running times.

- Based on 2 important ideas:
  - Residual Graphs (includes reverse edges)
  - Augmentation paths (path in a residual graph)

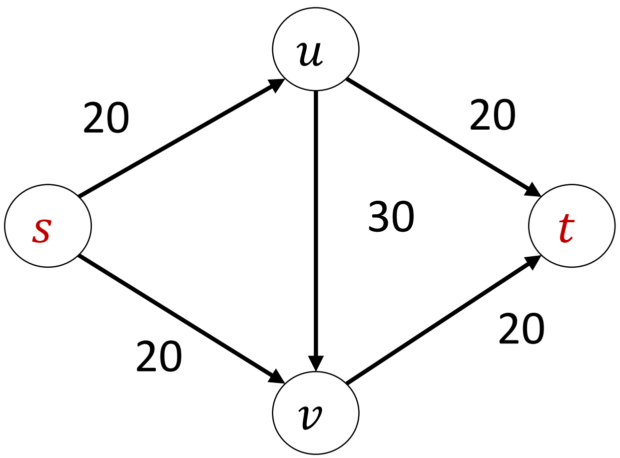
Residual Graph

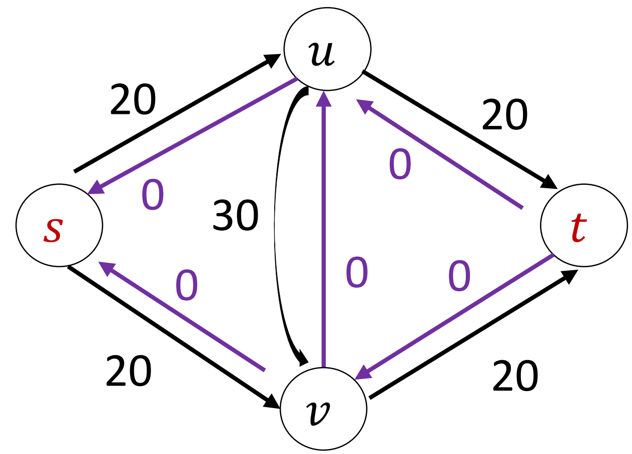
Given a network G and a flow f, we construct a residual graph G_f, representing places where flow can still be added to f, including places where existing flow can be decreased.

- G_f is defined as follows:
  - G_f contains same nodes a G.
  - Forward edges: for each edge $e = (u, v)$ of G for which $f_e < c_e$, include an edge $e' = (u, v)$ in G_f with capacity $c_e - f_e$
  - Backward edges (represent decreasing flow): for each edge $e = (u, v)$ of G with $f_e > 0$, include an edge $e' = (v, u)$ in G_f with capacity f_e

Residual network

- Initially flow is 0 for all edges.





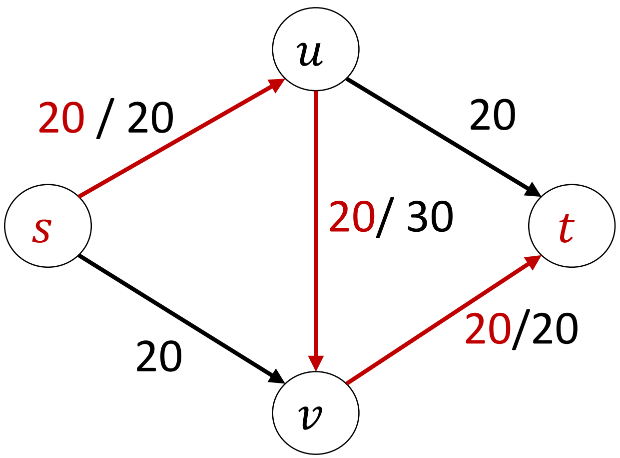
Residual network

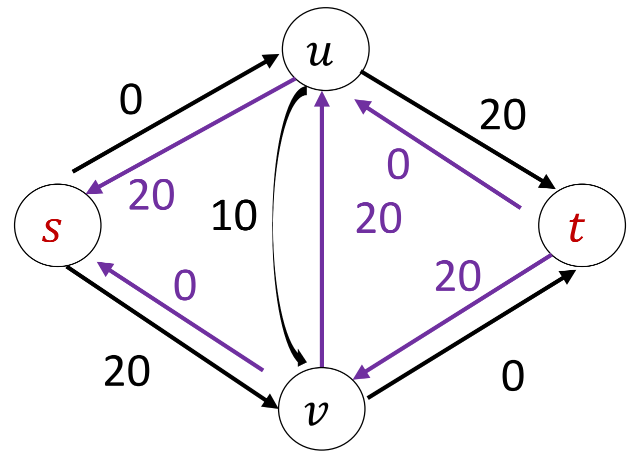
G_f contains same nodes a G.

Forward edges: for each edge $e = (u, v)$ of G for which $f_e < c_e$, include an edge $e' = (u, v)$ in G_f with capacity $c_e - f_e$

Backward edges: for each edge $e = (u, v)$ of G with $f_e > 0$, include an edge $e' = (v, u)$ in G_f with capacity f_e

Residual network



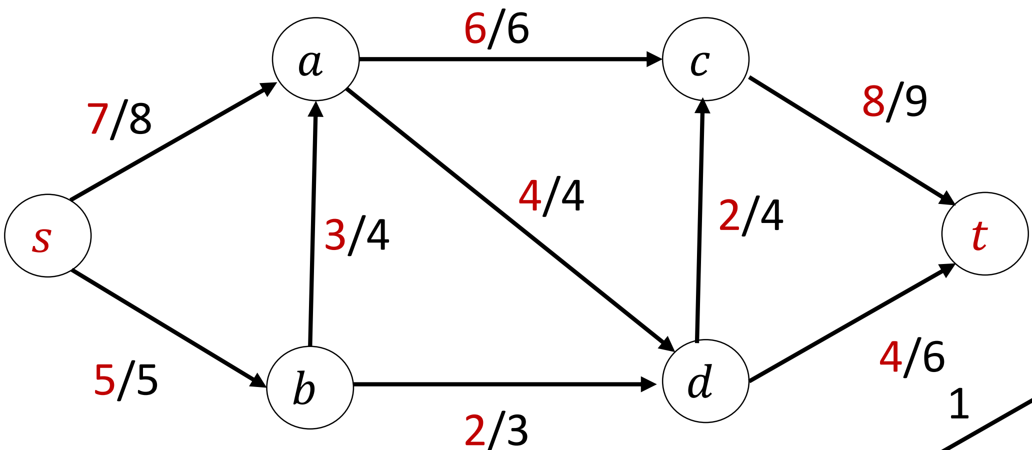


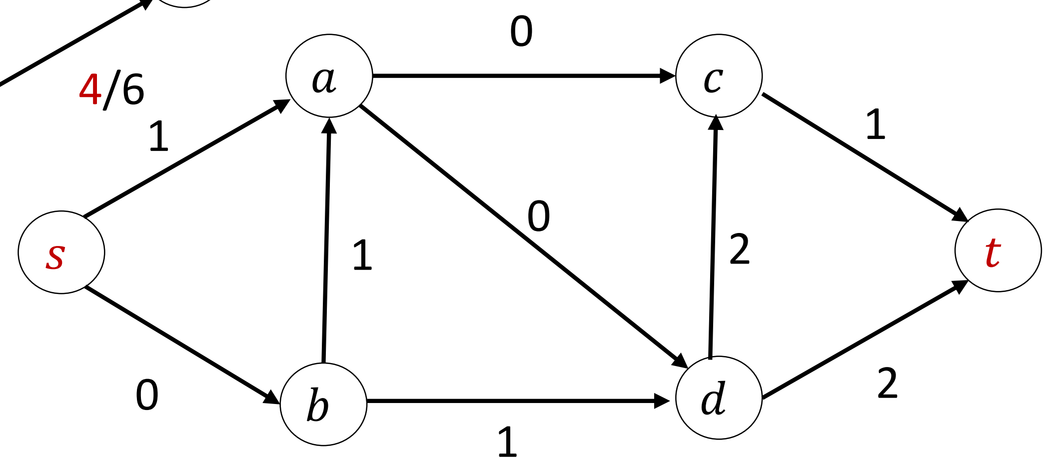
G_f contains same nodes a G.

Forward edges: for each edge $e = (u, v)$ of G for which $f_e < c_e$, include an edge $e' = (u, v)$ in G_f with capacity $c_e - f_e$

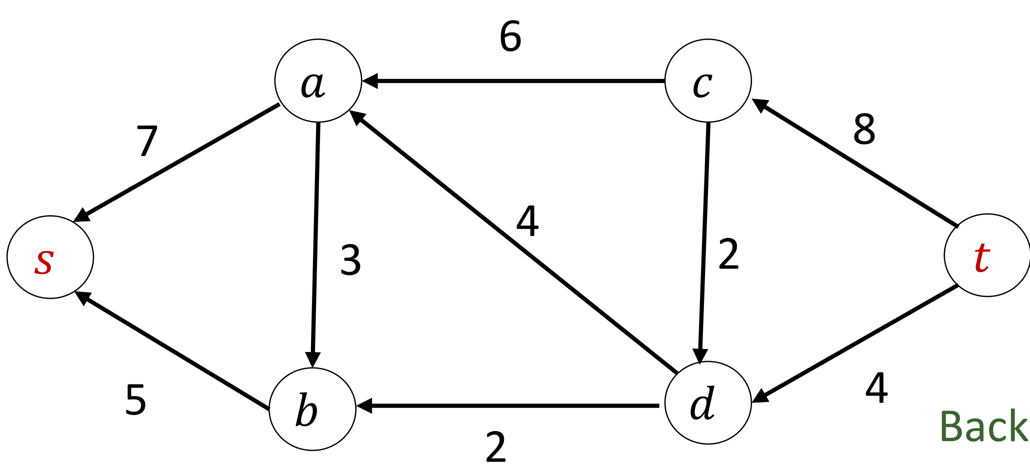
Backward edges: for each edge $e = (u, v)$ of G with $f_e > 0$, include an edge $e' = (v, u)$ in G_f with capacity f_e

Example



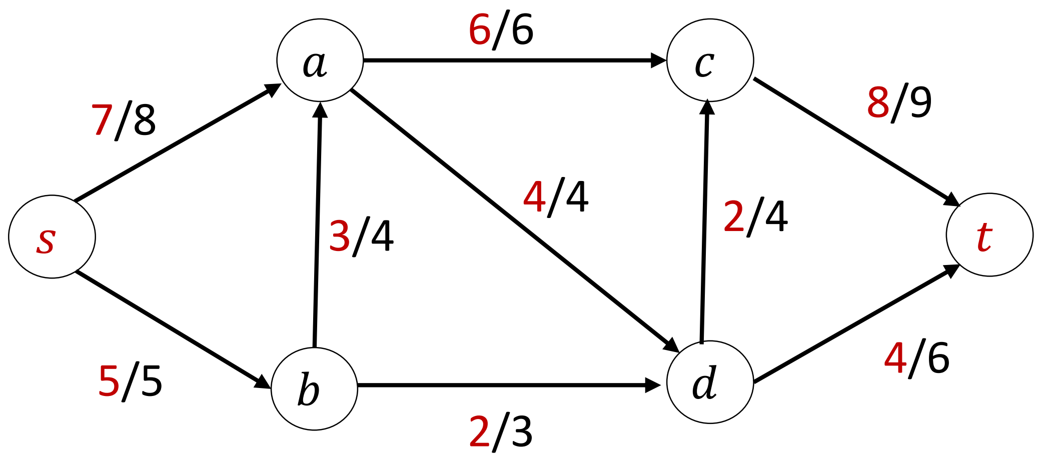


Forward edges

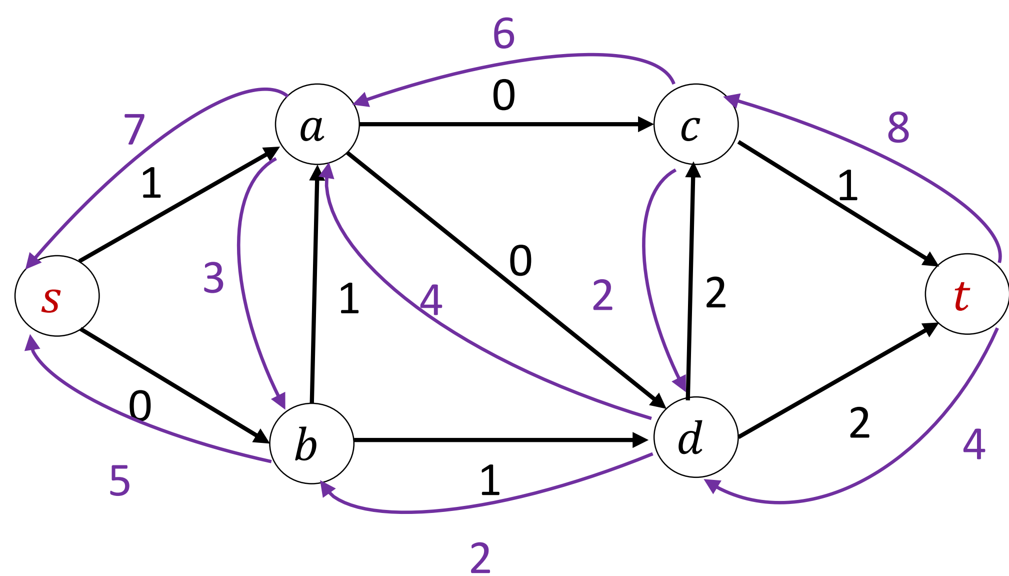


Backward edges

Example



a
c
s
t
b
d
6/6
7/8
8/9
4/4
3/4
2/4
5/5
4/6
2/3



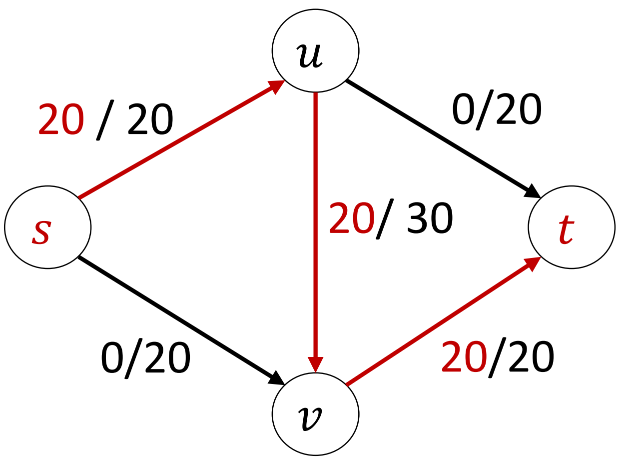
6
0
7
8
1
1
a
c
s
t
b
d
3
1
4
0
2
2
0
2
5
1
4
2
Residual network

Augmentation Path

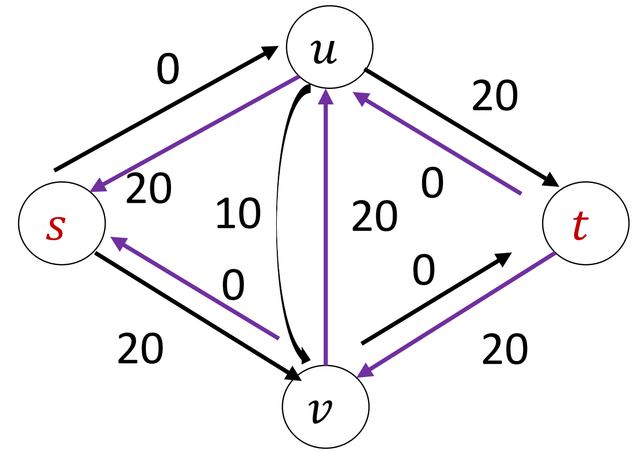
- Augmentation path: Given a network G and a flow f. Augmentation path is any flow g on residual graph G_f from source s to destination t.

- Augmentation path g can be added to f to get a new flow on G.
  - g_e (forward edge) adds to f_e
  - g'_e (backward edge) subtracts from f_e (equivalent of reversing our previous decision)

Example

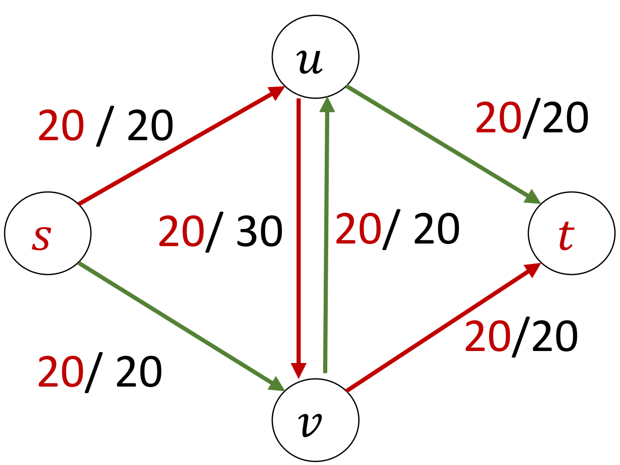


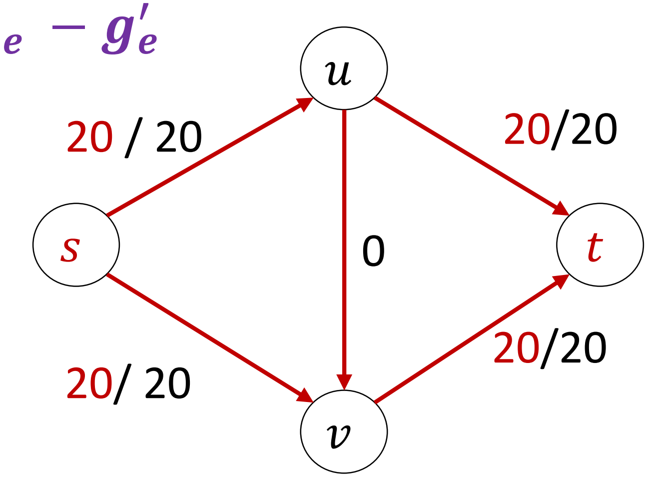
Residual Graph



Overall flow is given by:

$$f_e + g_e - g'_e$$

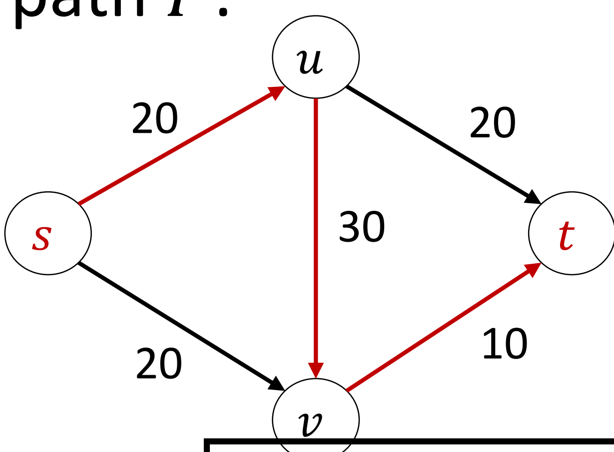


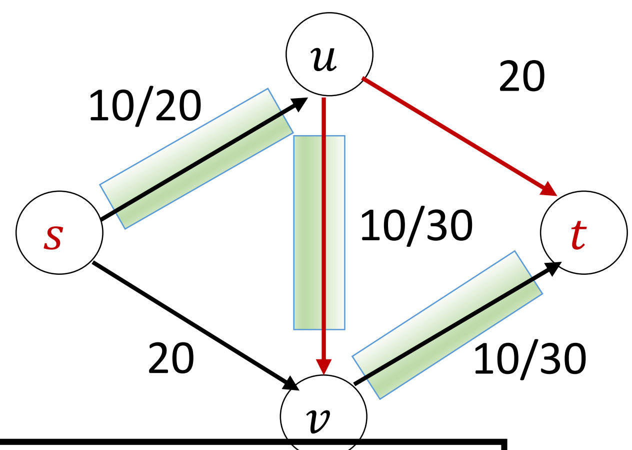


Augmenting Paths

- How much flow can be added in each step?

- Bottleneck (P, G_f) : the smallest capacity in G_f on any edge of P. If bottleneck $(P, G_f) > 0$ then we can increase the flow by sending bottleneck (P, G_f) along the path P.





Bottleneck $(P, G_f) = \min\{(c_e) : e \text{ is in } P\}$

Ford-Fulkerson method

- Follows a greedy approach. Iteratively increase the value of flow.

$\boldsymbol{FordFulkerson(G, s, t)}$

Let $f_e = 0$ for all edges (no flow anywhere)

Initialize Residual Graph RG // for every forward edge in the original graph G add a reverse edge with a capacity 0

$maxFlow = 0$

Repeat

Find some P path from s to t in RG such that $c_e > 0$ for all edges in P

if P exists

$pathFlow = Bottleneck\ (P, RG)$

$maxFlow = maxFlow + pathFlow$

Output $(P, pathflow)$ as an augmentation path

Update RG

Endif

Until the RG has no more augmentation paths.

Output $maxFlow$

Note: Ford-Fulkerson does not state how to find augmentation paths.

Updating Residual Graph

```
For each edge e = (u, v) ∈ P:
    f(u,v) = f(u,v) + pathFlow //add flow
    c(u,v) = c(u,v) − pathFlow //reduce capacity

    c(v,u) = c(v,u) + pathFlow//increase capacity in reverse edge
EndFor
```

Edmonds-Karp algorithm is an implementation of the Ford-Fulkerson method that uses BFS for finding augmenting paths.

```
BFS(RG, s, t)
Define augPath as ArrayList of edges
q := queue()
q.push(s)
backpointer(v) = null for all v  // backpointer data-structure to hold
                                 //edges that lead to the vertex
while !q.isEmpty()
   cur := q.pull()
   for each outedge e of cur in RG do
     if e.toCity! = s and backpointer(e.toCity) == null and e.cap ≠ 0
         backpointer(e.tocity):= e
         if(backpointer(t)! = null)  // found a path from s to t. Build it now from reverse
            pathEdge = backpointer(t)
            while(pathEdge ! = null)
                augPath.add(pathEdge)
                pathEdge = backpointer(pathEdge.fromCity)
            endWhile
            Collections.reverse(augPath)
            return augPath
         endIF
         q.push(e.toCity)
     endIf
   endFor
endWhile
return null
```

Next Lecture

- Example – Edmonds-Karp algorithm