CSL 356: Analysis and Design of Algorithms

Ragesh Jaiswal

CSE, IIT Delhi

Techniques

- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Network Flow
- Computational Intractability

Topics

- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Network Flow
- Computational intractability
- <u>Other topics</u>: Randomized algorithms, computational geometry, Number-theoretic algorithms etc.

Network Flow

- We want to model various kinds of networks using graphs and then solve real world problems w.r.t. these networks by studying the underlying graph.
- One problem that arises in network design is routing "flows" within the network.
  - Transportation network: Vertices are cities and edges denote highways. Every highway has certain traffic capacity. We are interested in knowing the maximum amount commodity that can be shipped from a source city to a destination city.
  - Computer network: edges are links and vertices are switches. Each link has some capacity of carrying packets. Again, we are interested in knowing how much traffic can a source node send to a destination node.

Network Flow

- To model these problems, we consider weighted, directed graph $G = (V, E)$ with the following properties:
  - *Capacity*: Associated with each edge e is a capacity that is a non-negative *integer* denoted by $c(e)$.
  - *Source node*: There is a source node s with no incoming edges.
  - *Sink node*: There is a sink node t with no outgoing edges.

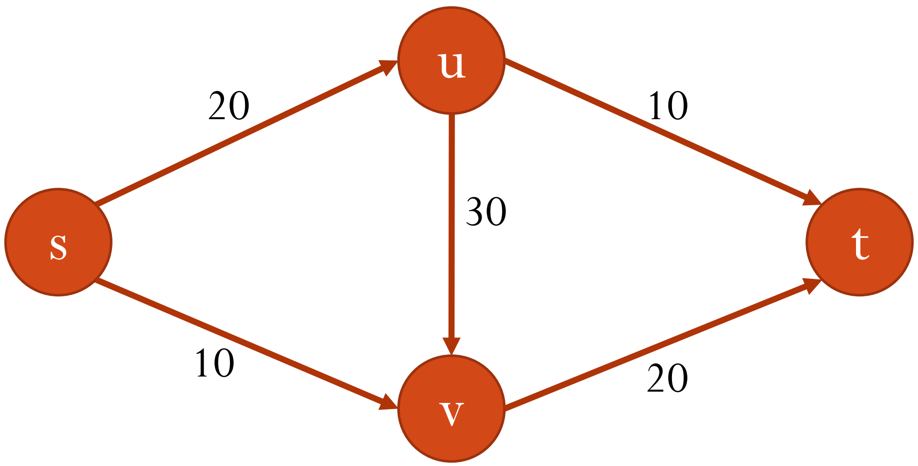
  All other nodes in the graph are called *internal nodes*.
- Given such a graph an "s-t flow" in the graph is a function f that maps the edges to non-negative real numbers such that the following properties are satisfied:
  - *Capacity constraint*: For all edges e, $0 \leq f(e) \leq c(e)$.
  - *Flow conservation*: For every internal node v,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e).$$

Network Flow

- Problem (maximum flow): Find a s-t flow f such that the following quantity is maximized:

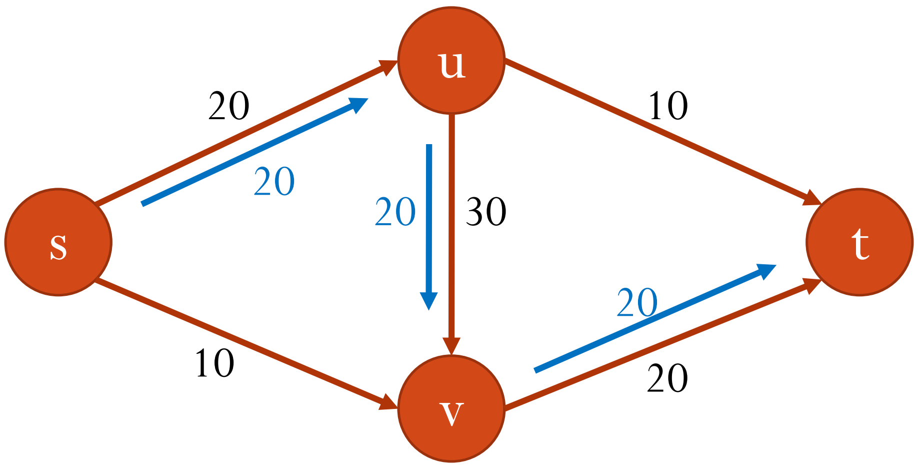
$$v(f) = \sum_{e \; out \; of \; s} f(e)$$



Network Flow

- <u>Problem (maximum flow)</u>: Find a s-t flow f such that the following quantity is maximized:

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

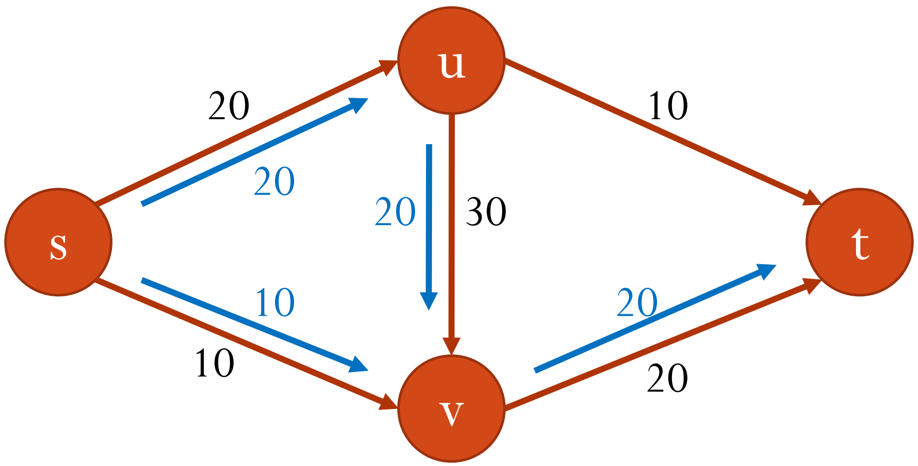


Routing 20 units of flow from **s** to **t**.
Is it possible to "push more flow"?

Network Flow

- Problem (maximum flow): Find a s-t flow f such that the following quantity is maximized:

$$v(f) = \sum_{e \ out \ of \ s} f(e)$$

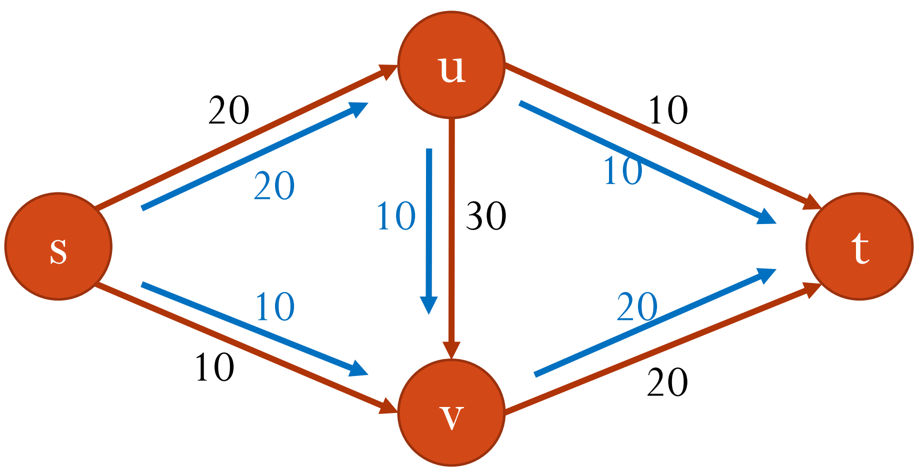


We should reset the initial flow (u, v) to 10

Network Flow

- <u>Problem (maximum flow)</u>: Find a s-t flow f such that the following quantity is maximized:

$$v(f) = \sum_{e\ out\ of\ s} f(e)$$

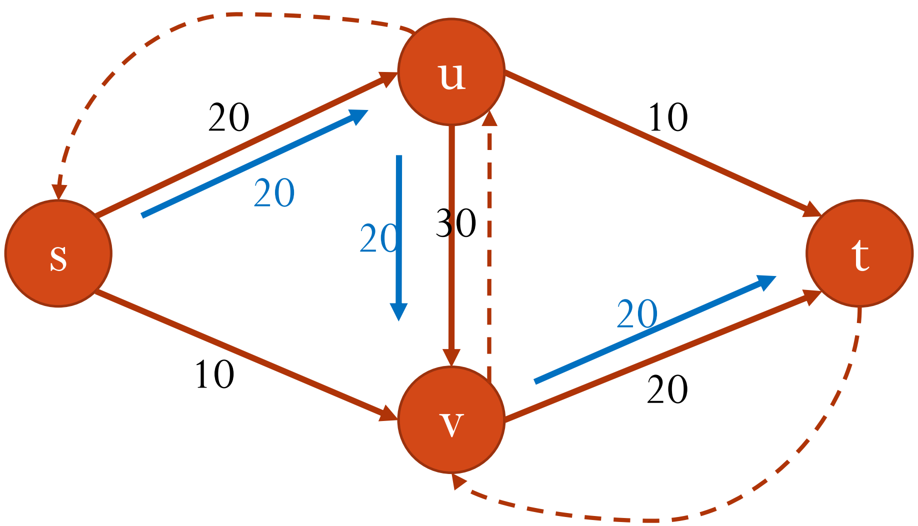


We should reset the initial flow (u, v) to 10

Maximum flow from $s\ =\ 30$

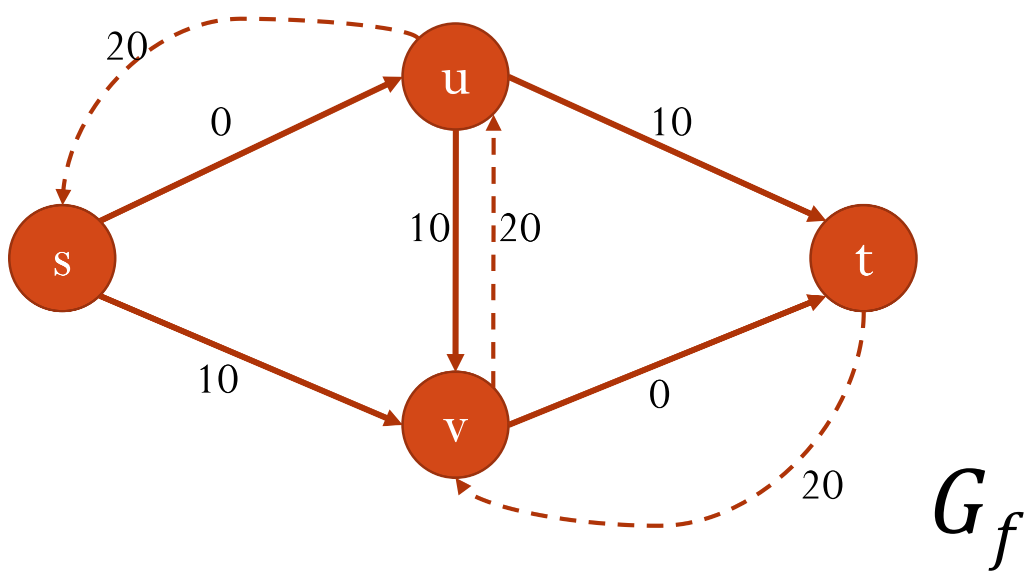
Network Flow

- Approach:
  - We will build *iteratively* build *larger* $s - t$ flows.
  - Given an $s - t$ flow f, we will build a *residual graph* G_f that will allow us to reset flows along some of the edges.
  - We will find an *augmenting path* in the residual graph G_f, push some flow along this path and update the flow to f'.



Network Flow

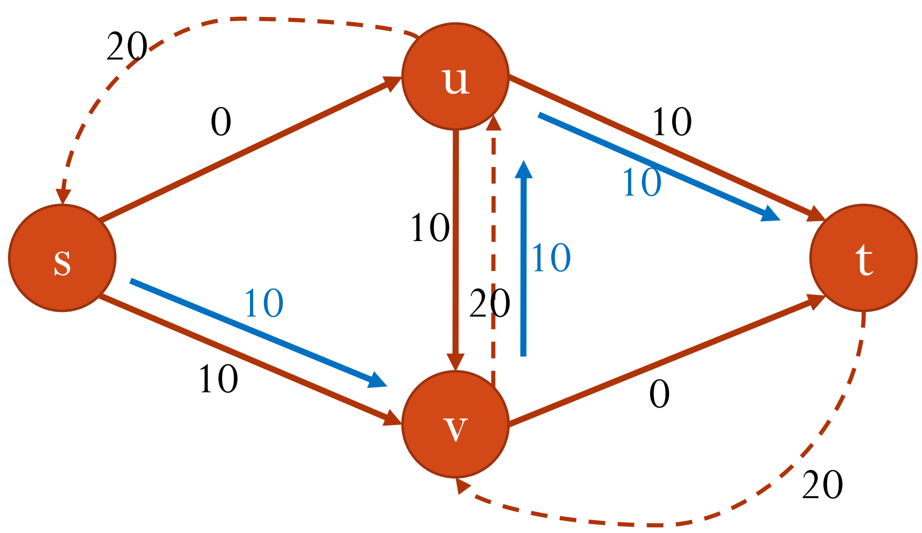
- <u>Approach</u>:
  - We will build *iteratively* build *larger* $s-t$ flows.
  - Given an $s-t$ flow f, we will build a *residual graph* G_f that will allow us to reset flows along some of the edges.
  - We will find an *augmenting path* in the residual graph G_f, push some flow along this path and update the flow to f'.



$f(s,u) = 20,$
$f(s,v) = 0$
$f(u,v) = 20,$
$f(u,t) = 0,$
$f(v,t) = 20$

Network Flow

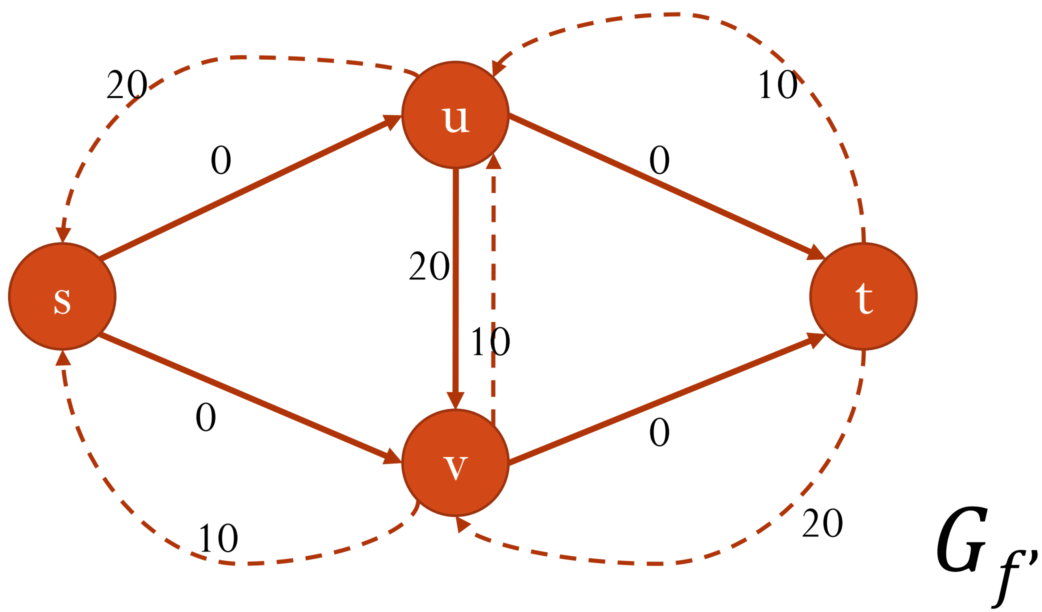
- <u>Approach</u>:
  - We will build *iteratively* build *larger* $s - t$ flows.
  - Given an $s - t$ flow f, we will build a *residual graph* G_f that will allow us to reset flows along some of the edges.
  - We will find an *augmenting path* in the residual graph G_f, push some flow along this path and update the flow to f'.



$f'(s,u) = 20,$
$f'(s,v) = 10$
$f'(u,v) = 10,$
$f'(u,t) = 10,$
$f'(v,t) = 20$

Network Flow

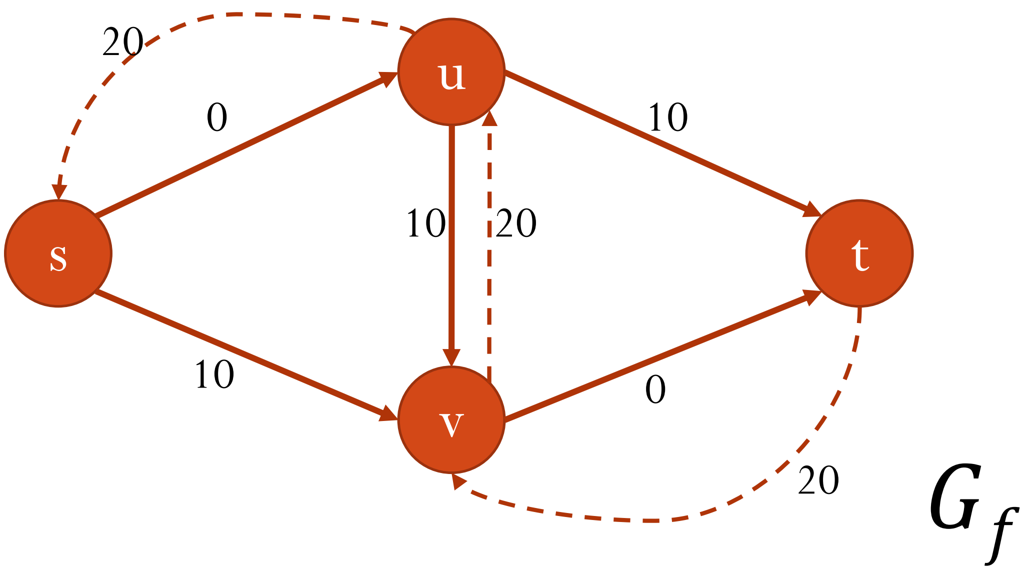
- Approach:
  - We will build *iteratively* build *larger* $s - t$ flows.
  - Given an $s - t$ flow f, we will build a *residual graph* G_f that will allow us to reset flows along some of the edges.
  - We will find an *augmenting path* in the residual graph G_f, push some flow along this path and update the flow to f'.



$f'(s,u) = 20,$
$f'(s,v) = 10$
$f'(u,v) = 10,$
$f'(u,t) = 10,$
$f'(v,t) = 20$

Network Flow

- <u>Residual Graph G_f:</u>
  - *Forward edges*: For every edge e in the original graph, there are $(c(e) - f(e))$ units of more flow we can send along that edge. So we set the weight of this edge to $(c(e) - f(e))$.
  - *Backward edges*: For every edge $e = (u, v)$ in the original graph, there are $f(e)$ units of flow that we can undo. So we add a reverse edge $e' = (v, u)$ and set the weight of e' to $f(e)$.

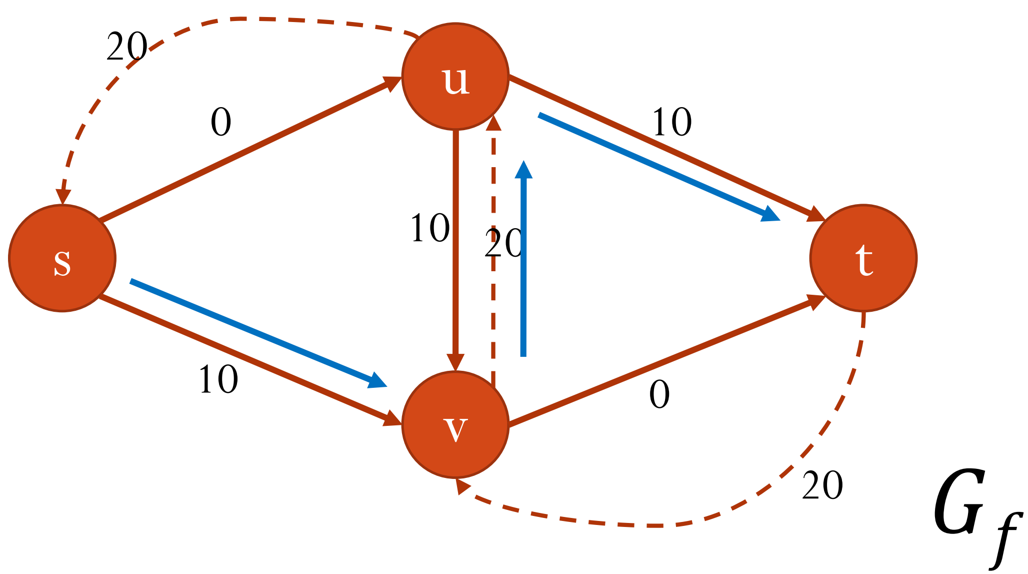


$f(s,u) = 20,$
$f(s,v) = 0$
$f(u,v) = 20,$
$f(u,t) = 0,$
$f(v,t) = 20$

Network Flow

- <u>Augmenting paths in G_f:</u>
  - Let P be a simple s-t path in G_f. Note that this contains forward and backward edges.
  - Let e_{min} be an edge in the path P of minimum weight w_{min}.
  - For every forward edge e in path P, set $f'(e) = f(e) + w_{min}$
  - For every backward edge (u, v) in P, set

$$f'(v, u) = f(v, u) - w_{min}$$

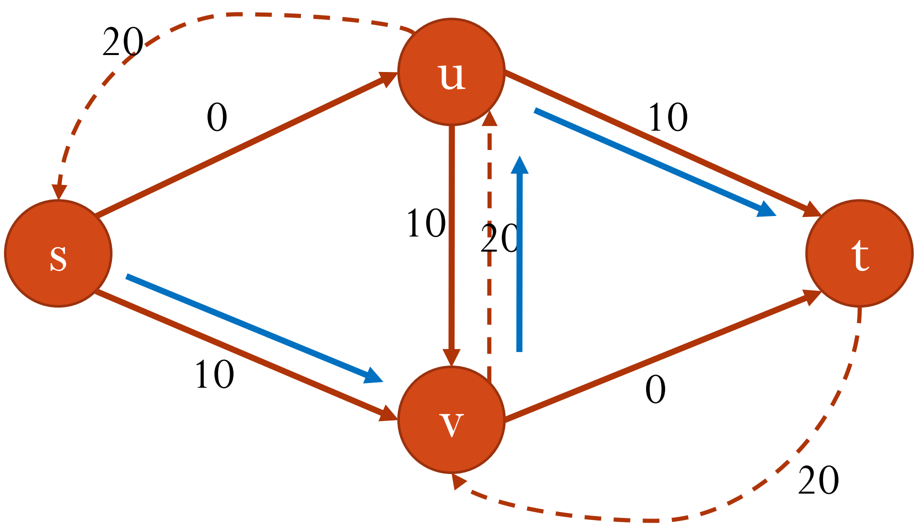


$f(s, u) = 20,$
$f(s, v) = 0$
$f(u, v) = 20,$
$f(u, t) = 0,$
$f(v, t) = 20$

Network Flow

- <u>Augmenting paths in G_f:</u>
  - Let P be a simple s-t path in G_f. Note that this contains forward and backward edges.
  - Let e_{min} be an edge in the path P of minimum weight w_{min}.
  - For every forward edge e in path P, set $f'(e) = f(e) + w_{min}$
  - For every backward edge (u, v) in P, set

$$f'(v, u) = f(v, u) - w_{min}$$

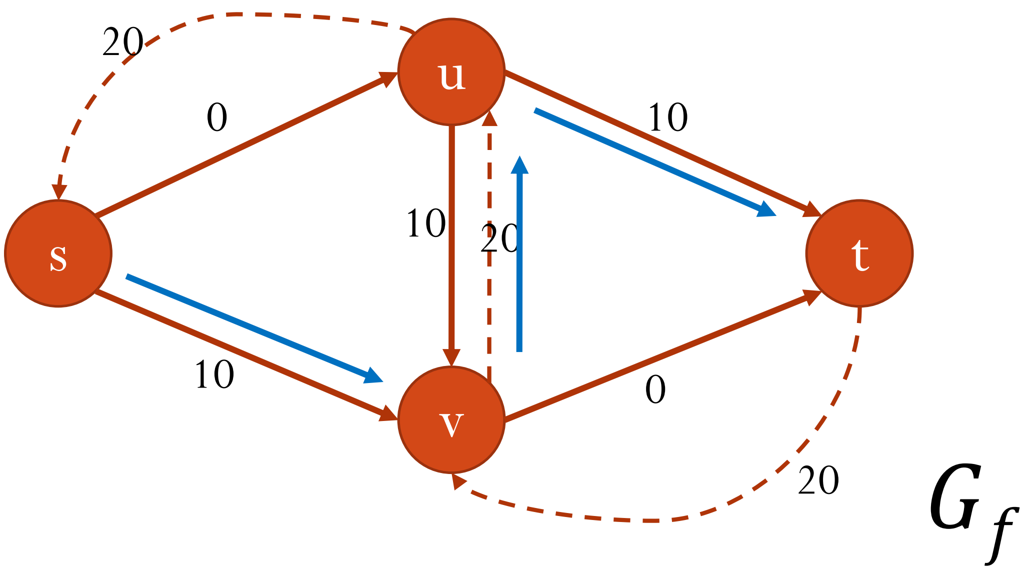


G_f

$f'(s, u) = 20,$
$f'(s, v) = 10$
$f'(u, v) = 10,$
$f'(u, t) = 10,$
$f'(v, t) = 20$

Network Flow

- Claim: f' is an $s-t$ flow.
- Proof:
  - Check capacity constraint for each edge.
  - Check flow conservation at each vertex.



$$f'(s,u) = 20,\\ f'(s,v) = 10\\ f'(u,v) = 10,\\ f'(u,t) = 10,\\ f'(v,t) = 20$$

Network Flow

Max-Flow *//Ford-Fulkerson algorithm*

- Start with a flow f such that $f(e) = 0$
- while there is an $s - t$ path P in G_f
  - Execute the augmenting path algorithm to obtain f'
  - Update f to f' and G_f to $G_{f'}$
- return f

- Running time:

Network Flow

Max-Flow *//Ford-Fulkerson algorithm*

- Start with a flow f such that $f(e) = 0$
- while there is an $s - t$ path P in G_f
  - Execute the augmenting path algorithm to obtain f'
  - Update f to f' and G_f to $G_{f'}$
- return f

- Running time:
  - Claim 1: $v(f') > v(f)$.

Network Flow

Max-Flow *//Ford-Fulkerson algorithm*

- Start with a flow f such that $f(e) = 0$
- while there is an $s - t$ path P in G_f
  - Execute the augmenting path algorithm to obtain f'
  - Update f to f' and G_f to $G_{f'}$
- return f

- Running time:
  - Claim 1: $v(f') > v(f)$.
  - Claim 2: The while loop runs for $C = \sum_{e \text{ out of } s} c(e)$ iterations.

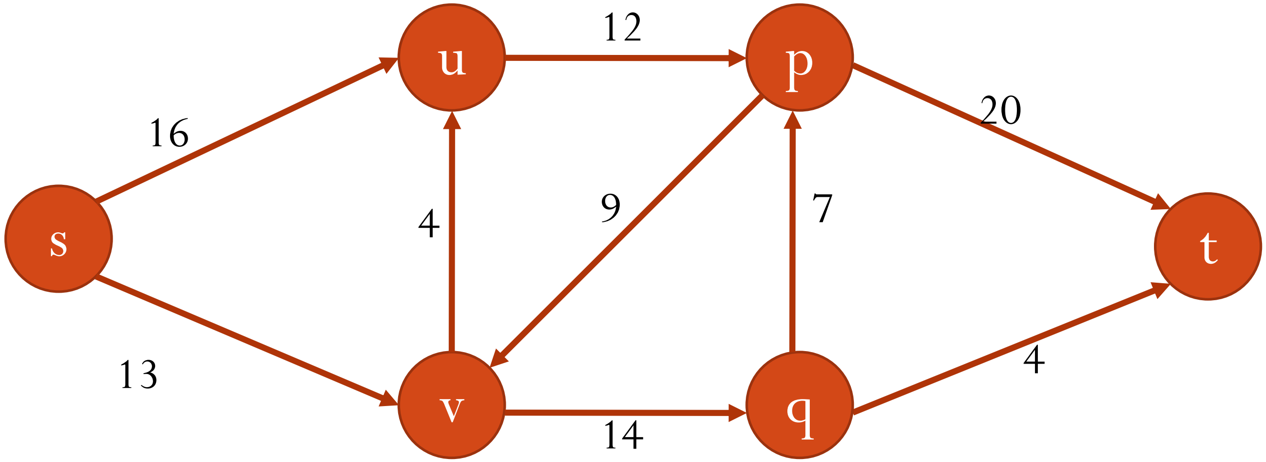
Network Flow

Max-Flow //*Ford-Fulkerson algorithm*

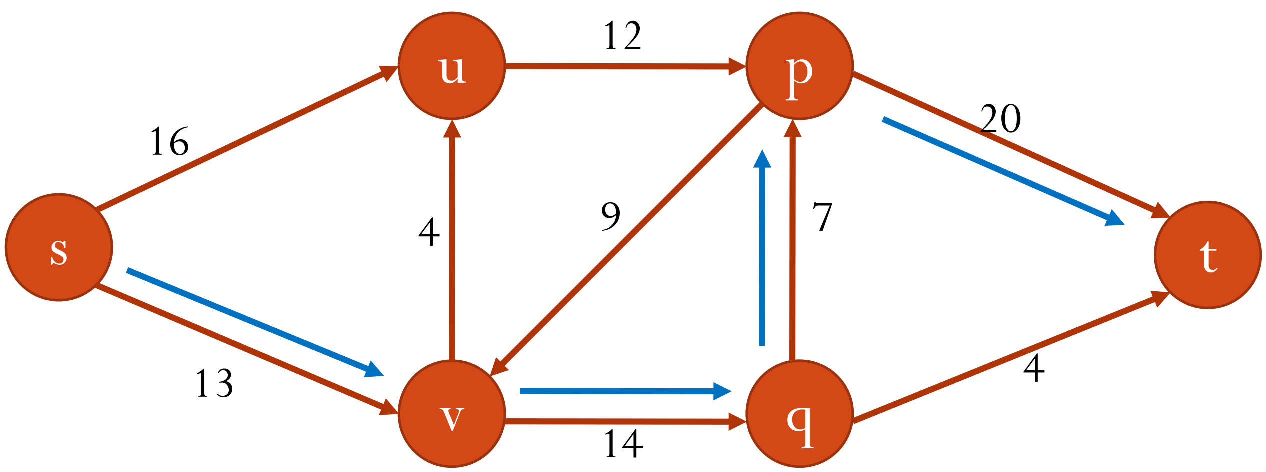
- Start with a flow f such that $f(e) = 0$
- while there is an $s - t$ path P in G_f
    - Execute the augmenting path algorithm to obtain f'
    - Update f to f' and G_f to $G_{f'}$
- return f

- Running time: $O(m \cdot C)$
    - Claim 1: $v(f') > v(f)$.
    - Claim 2: The while loop runs for $C = \sum_{e \text{ out of } s} c(e)$ iterations.
    - Claim 3: Augmenting a path takes $O(m)$ time

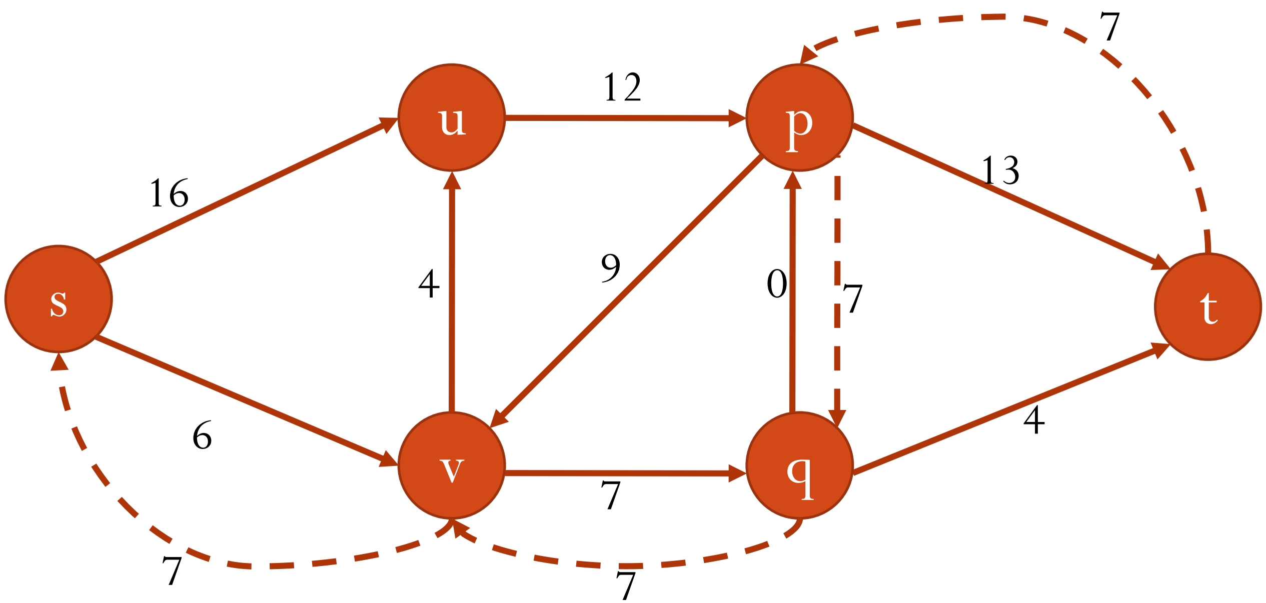
Network Flow



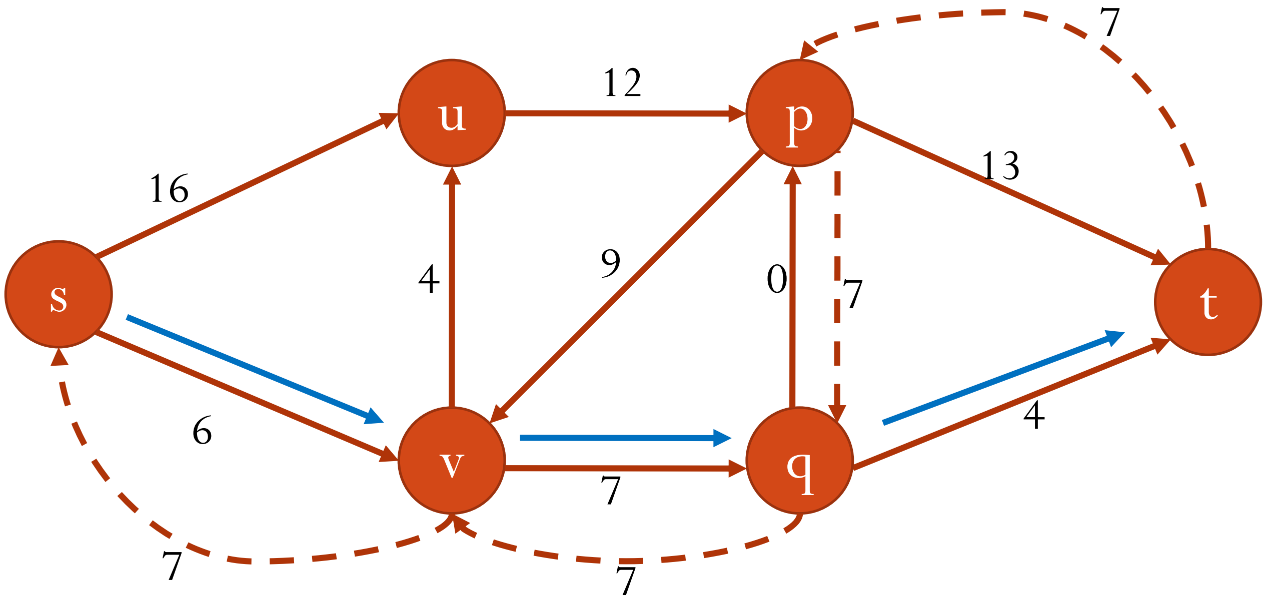
Network Flow



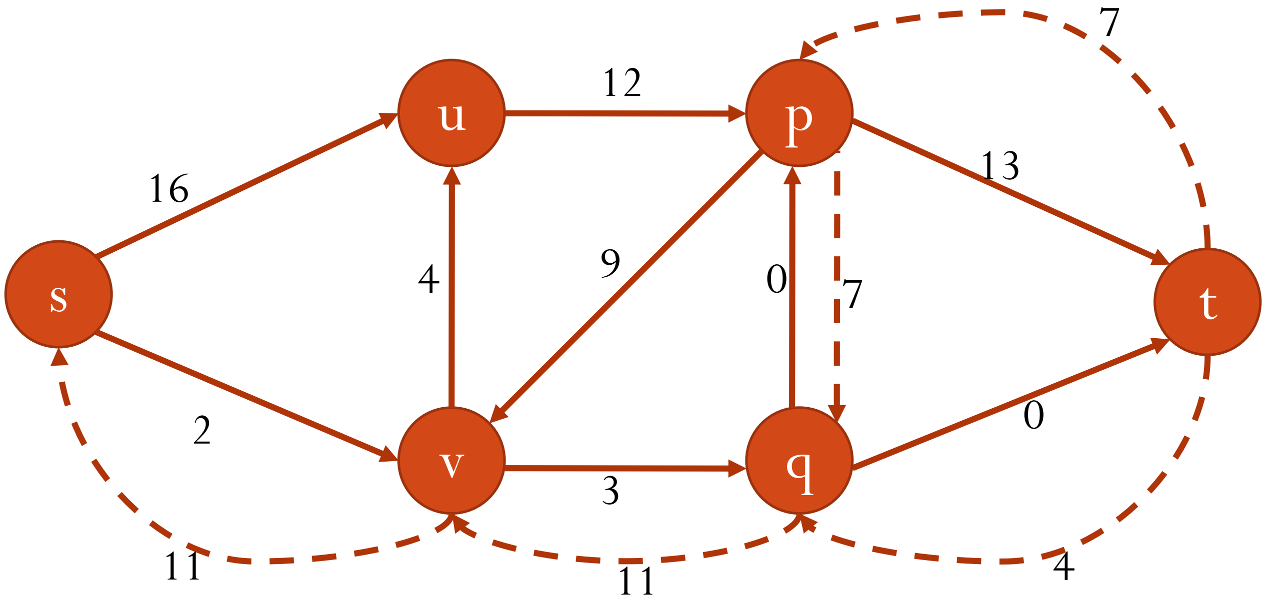
Network Flow



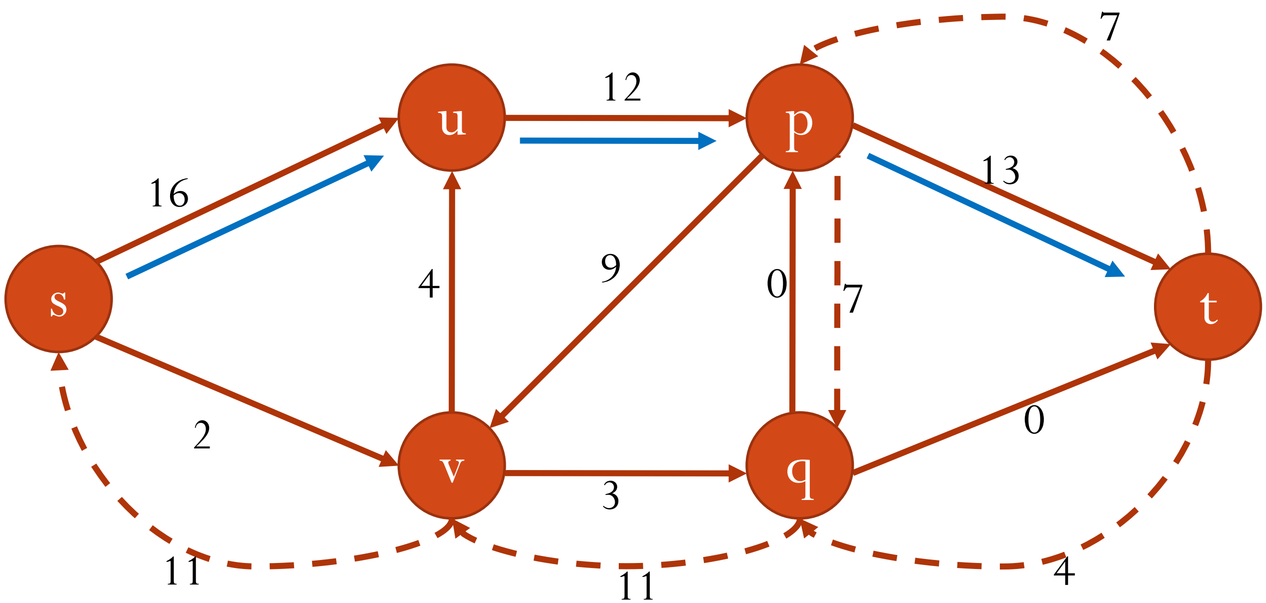
Network Flow



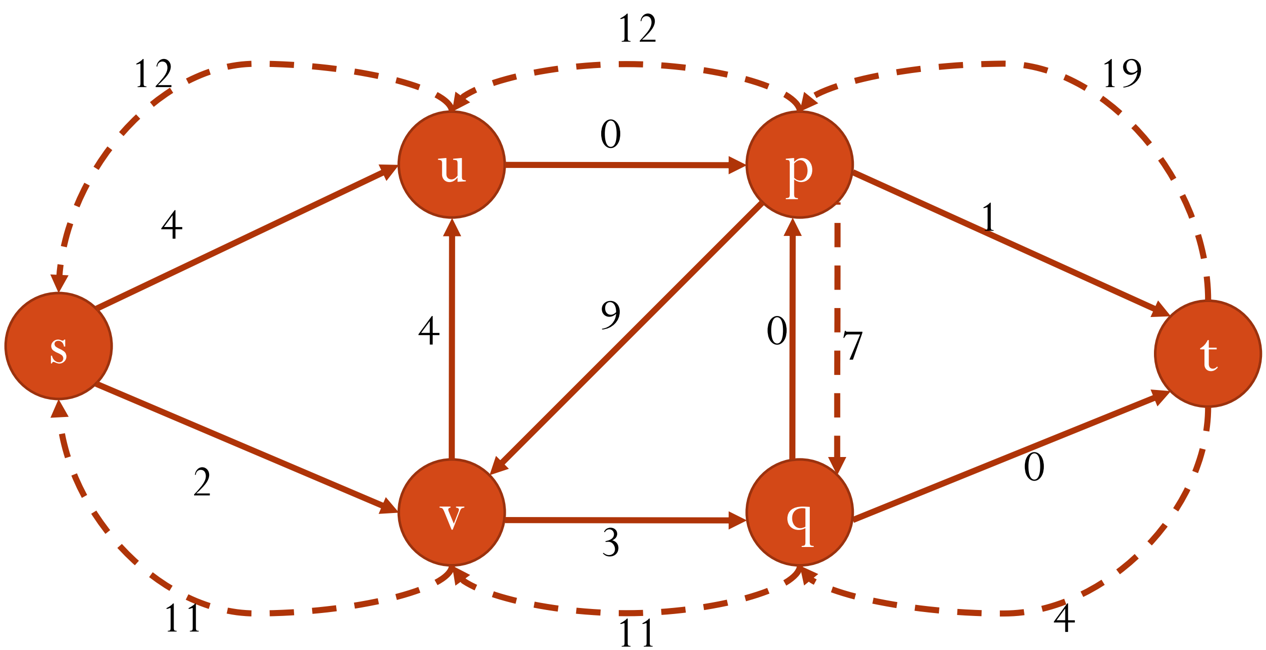
Network Flow



Network Flow



Network Flow



End

Problems to think about:

1. Consider the Ford-Fulkerson algorithm. Given an $s-t$ flow f, the algorithm picks an arbitrary $s-t$ path and pushes more flow along that path. Suppose we change the algorithm slightly and instead of picking an arbitrary $s-t$ path, pick a path with shortest hop-length (do a BFS and pick a shortest path). Can you construct an example where this algorithm will perform much better than the Ford-Fulkerson algorithm.