# COL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

# Course Overview

- Graph Algorithms
- Algorithm Design Techniques:
  - Greedy Algorithms
  - Divide and Conquer
  - Dynamic Programming
  - Network Flows
- Computational Intractability

# Network Flow

- Main Idea: Reduction
  1. We will obtain an algorithm $A$ for a *Network Flow* problem.
  2. Given a new problem, we will *rephrase* this problem as a Network Flow problem.
  3. We will then use algorithm $A$ to solve the rephrased problem and obtain a solution.
  4. Finally, we build a solution for the original problem using the solution to the rephrased problem.

- We want to model various kinds of networks using graphs and then solve real world problems with respect to 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 Networks: 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.

- 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 in-coming edges.
  - Sink node: There is a sink node $t$ with no out-going edges.

  All other nodes are called *internal nodes*.

- 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 in-coming edges.
  - Sink node: There is a sink node $t$ with no out-going edges.
    All other nodes 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:
  (a) Capacity constraint: For every edge $e$, $0 \leq f(e) \leq c(e)$.
  (b) Flow conservation: For every internal node $v$:

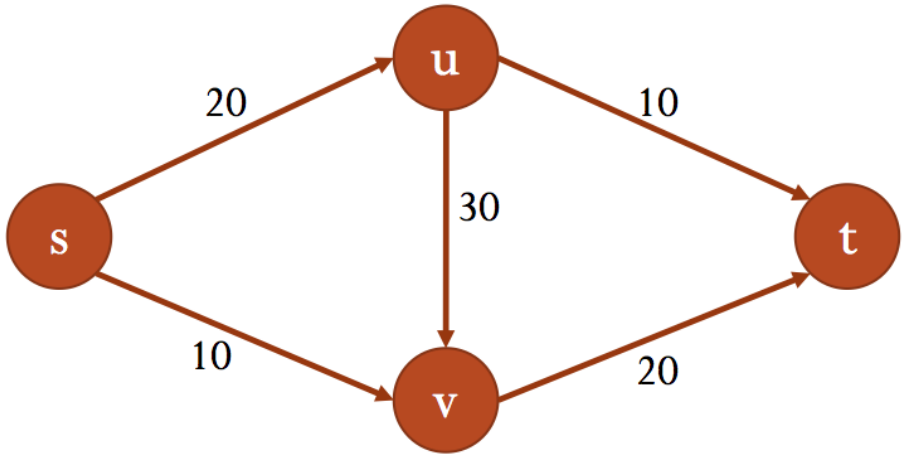
$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

## Problem

Find an $s - t$ flow $f$ in a given network graph such that the following quantity is maximized:

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

- Example:



**Problem**

Find an $s - t$ flow $f$ in a given network graph such that the following quantity is maximized:

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

- Example:

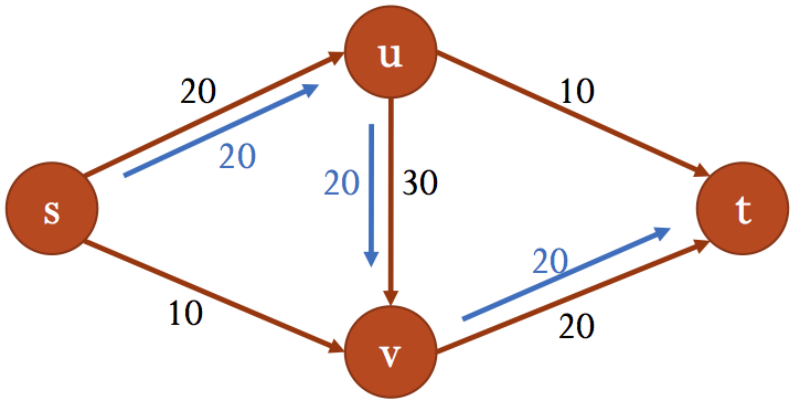



Figure: Routing 20 units of flow from $s$ to $t$. Is it possible to "push more flow"?

**Problem**

Find an $s-t$ flow $f$ in a given network graph such that the following quantity is maximized:

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

- Example:

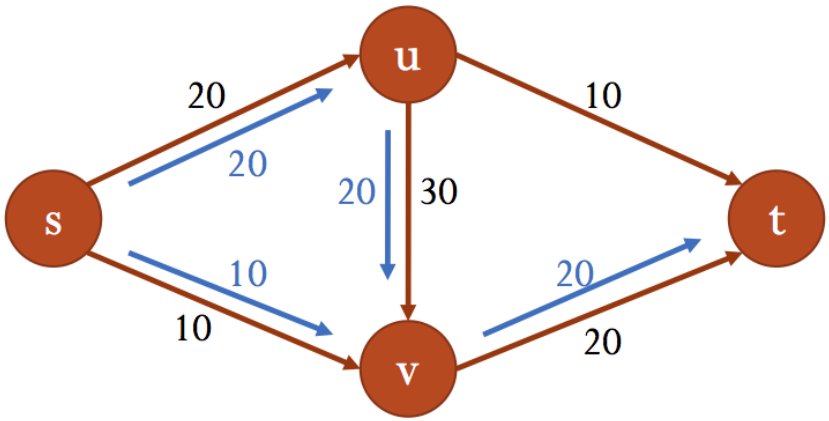



Figure: We should reset initial flow $(u, v)$ to 10.

**Problem**

Find an $s-t$ flow $f$ in a given network graph such that the following quantity is maximized:

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

- Example:

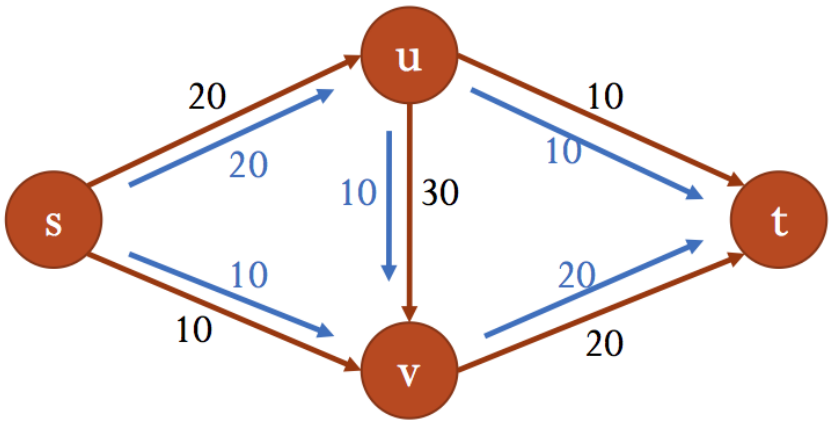
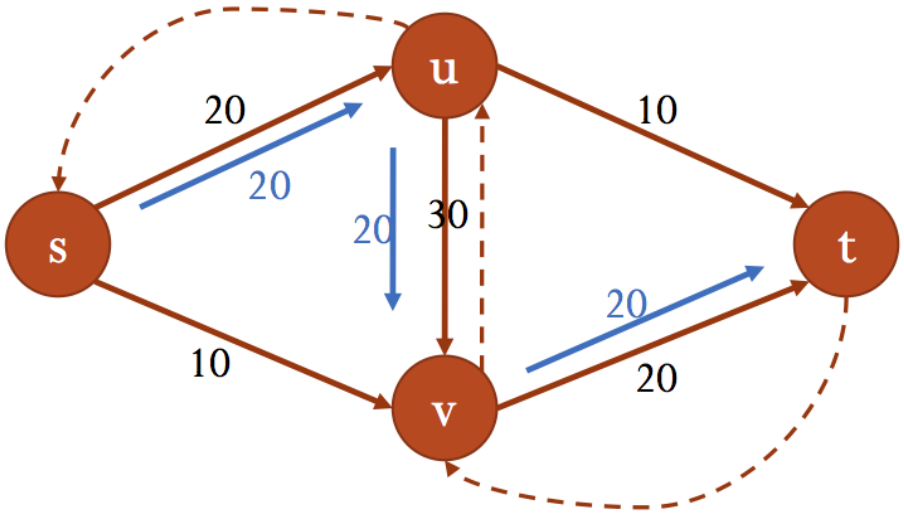



Figure: We should reset initial flow $(u, v)$ to 10. Maximum flow from $s$ is 30.

## Approach

- We will 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 $f'$.



## Approach

- We will 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 $f'$.

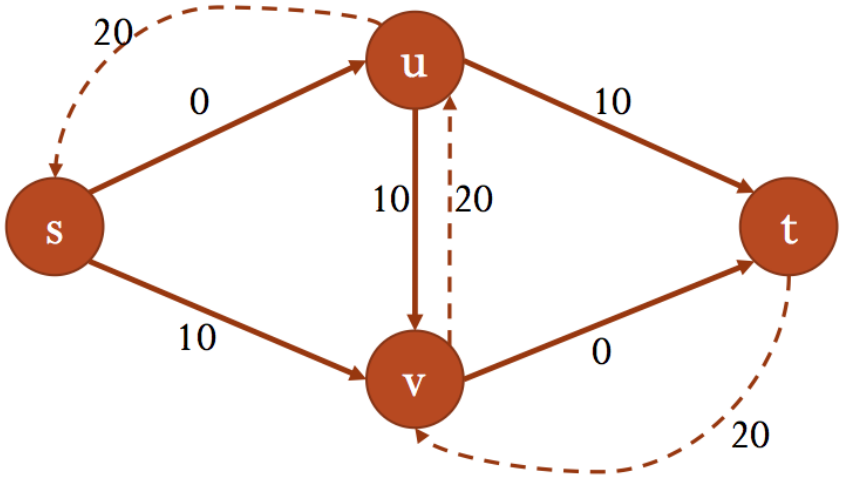



Figure: Graph $G_f$. *(f(s, u) = 20, f(s, v) = 0, f(u, v) = 20, f(u, t) = 0, f(v, t) = 20)*

## Approach

- We will 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 $f'$.

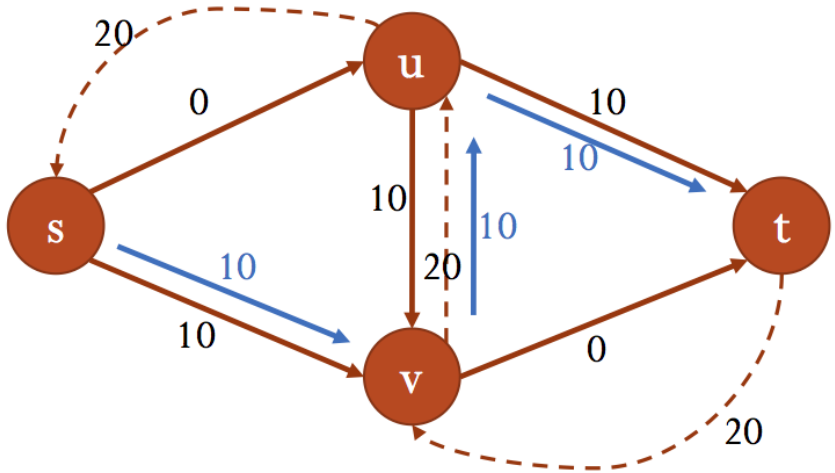



Figure: Augmenting path. *(f'(s, u) = 20, f'(s, v) = 10, f'(u, v) = 10, f'(u, t) = 10, f'(v, t) = 20)*

## Approach

- We will 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 $f'$.

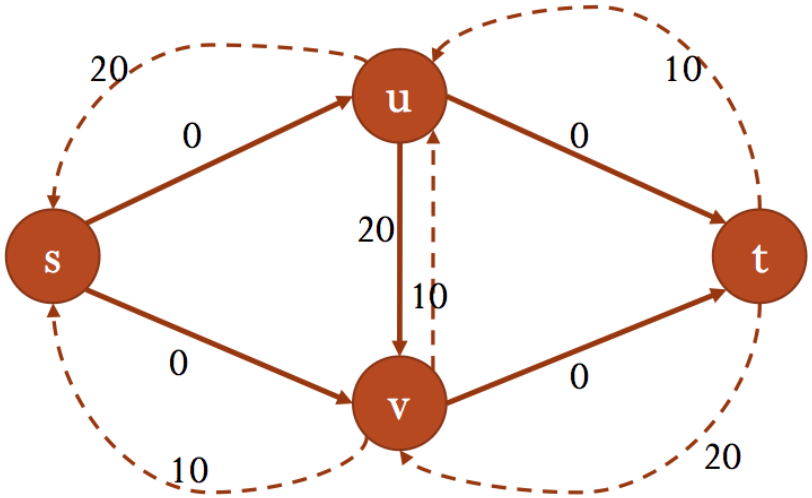



Figure: Graph $G_{f'}$. *(f'(s, u) = 20, f'(s, v) = 10, f'(u, v) = 10, f'(u, t) = 10, f'(v, t) = 20)*

- Residual graph $G_f$:
  - 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 $(c(e) - f(e))$.
  - 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 $f(e)$.

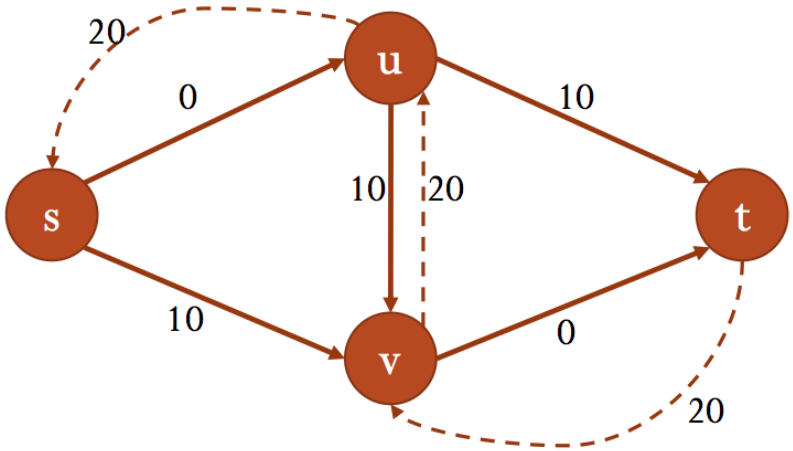



Figure: Graph $G_f$. *(f(s, u) = 20, f(s, v) = 0, f(u, v) = 20, f(u, t) = 0, f(v, t) = 20)*

- Augmenting flow in $G_f$:
  - 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 $P$ with minimum weight $w_{min}$
  - For every forward edge $e$ in $P$, set $f'(e) \leftarrow f(e) + w_{min}$
  - For every backward edge $(x, y)$ in $P$, set $f'(y, x) \leftarrow f(y, x) - w_{min}$
  - For all remaining edges $e$, $f'(e) = f(e)$

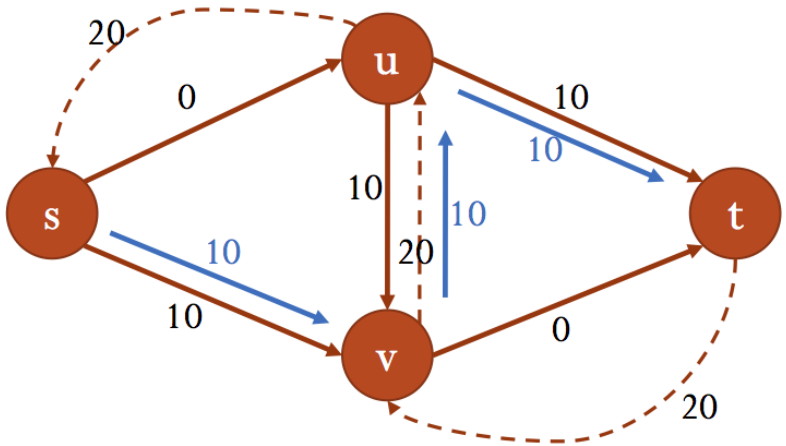



Figure: Augmenting path. *(f'(s, u) = 20, f'(s, v) = 10, f'(u, v) = 10, f'(u, t) = 10, f'(v, t) = 20)*

- Claim 1: $f'$ is an $s - t$ flow.
- Proof sketch:
  - Capacity constraint for each edge is satisfied.
  - Flow conservation at each vertex is satisfied.

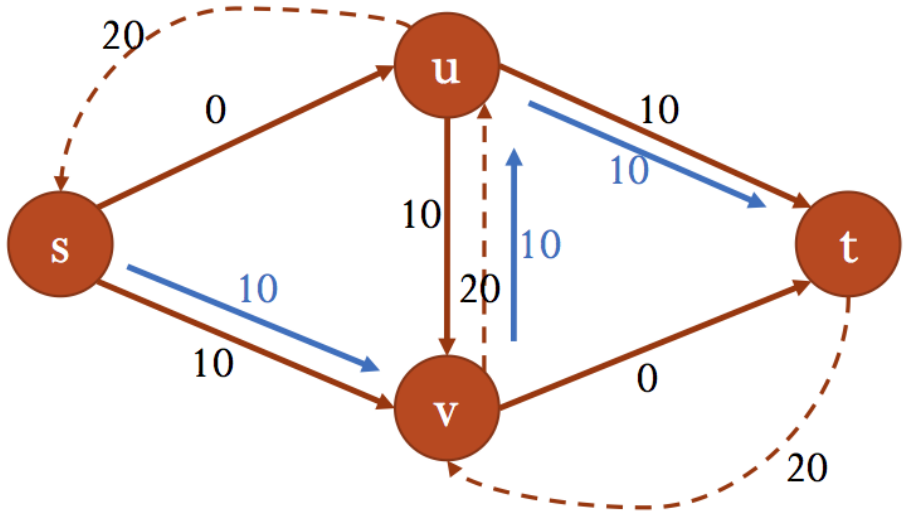



Figure: Augmenting path. *(f'(s, u) = 20, f'(s, v) = 10, f'(u, v) = 10, f'(u, t) = 10, f'(v, t) = 20)*

## Algorithm

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)

- What is the running time of the above algorithm?

## Algorithm

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)

- What is the running time of the above algorithm?
  - Claim 2: $v(f') > v(f)$.

## Algorithm

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)

- What is the running time of the above algorithm?
  - Claim 2: $v(f') > v(f)$.
  - Claim 3: The while loop runs for $C = \sum_{e \text{ out of } s} c(e)$ iterations.

## Algorithm

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)

- What is the running time of the above algorithm?
  - Claim 2: $v(f') > v(f)$.
  - Claim 3: The while loop runs for $C = \sum_{e \text{ out of } s} c(e)$ iterations.
  - Claim 4: Finding augmenting path and augmenting flow along this path takes $O(m)$ time.

## Algorithm

`Ford-Fulkerson`

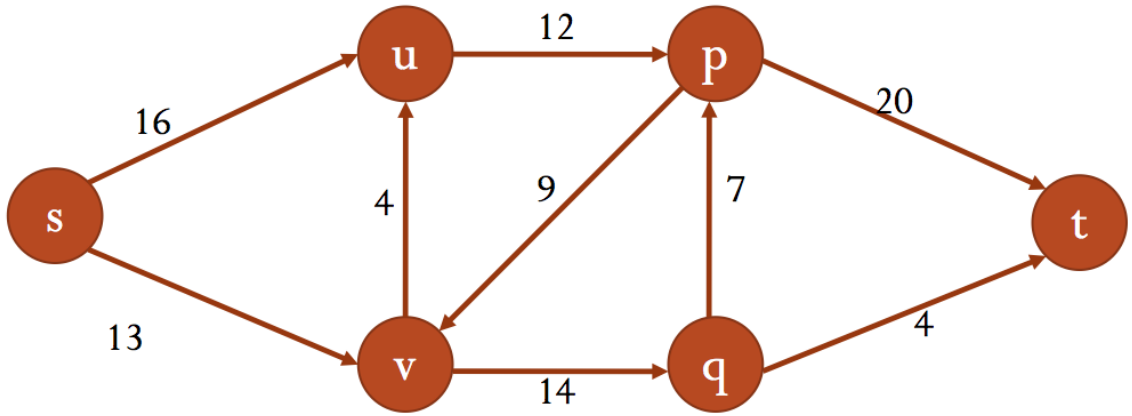
- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)

- What is the running time of the above algorithm? $O(m \cdot C)$
  - Claim 2: $v(f') > v(f)$.
  - Claim 3: The while loop runs for $C = \sum_{e \text{ out of } s} c(e)$ iterations.
  - Claim 4: Finding augmenting path and augmenting flow along this path takes $O(m)$ time.

## Algorithm

`Ford-Fulkerson`

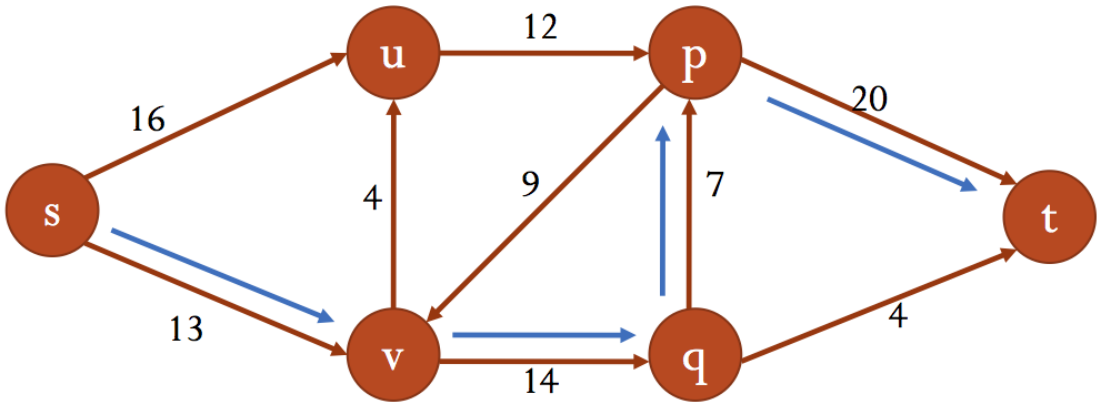
- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)



## Algorithm

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)



**Algorithm**

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)

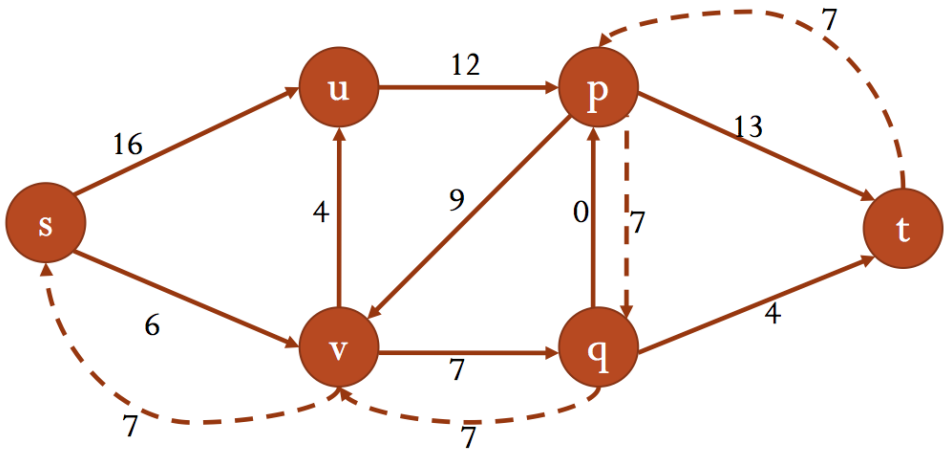
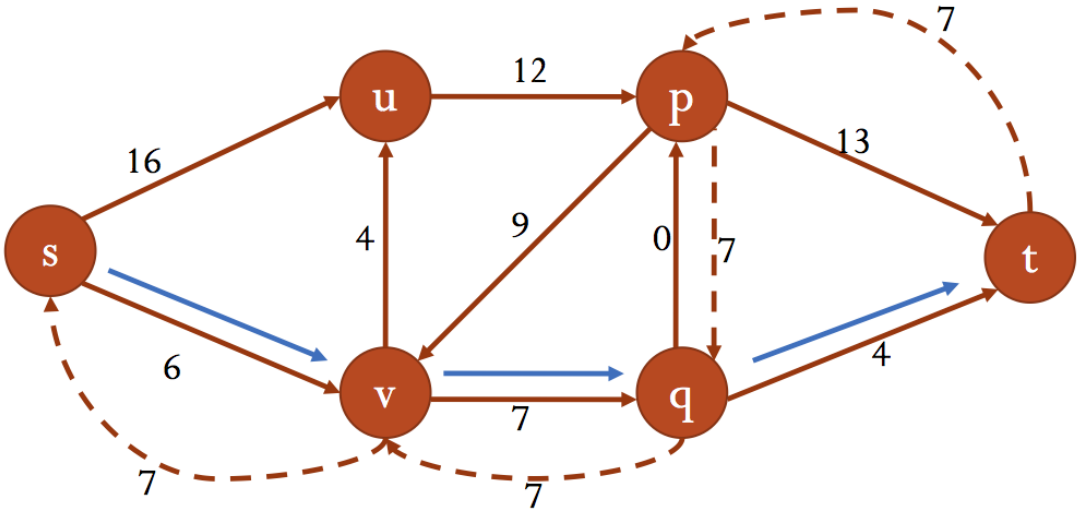



Figure: Graph $G_f$, where $f(s,u) = 0, f(s,v) = 7, f(v,u) = 0, f(v,q) = 7, f(u,p) = 0, f(p,v) = 0, f(p,t) = 7, f(q,p) = 7, f(q,t) = 0$

## Algorithm

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)



**Algorithm**

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)

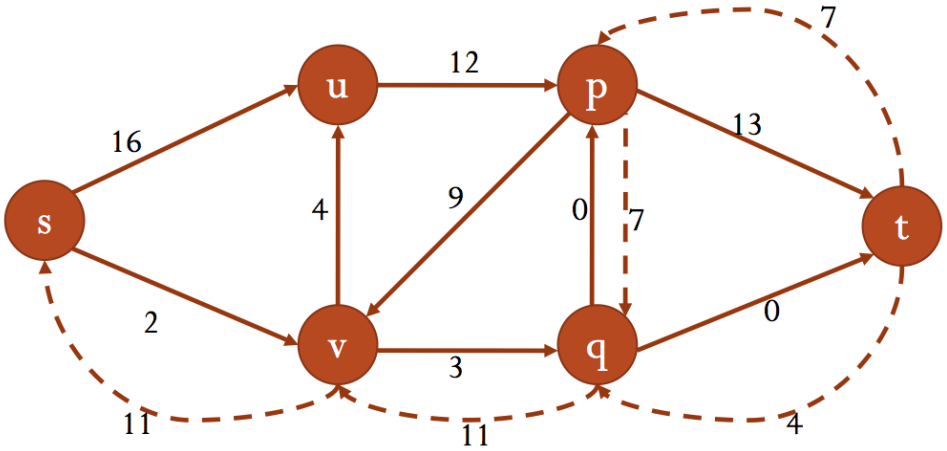
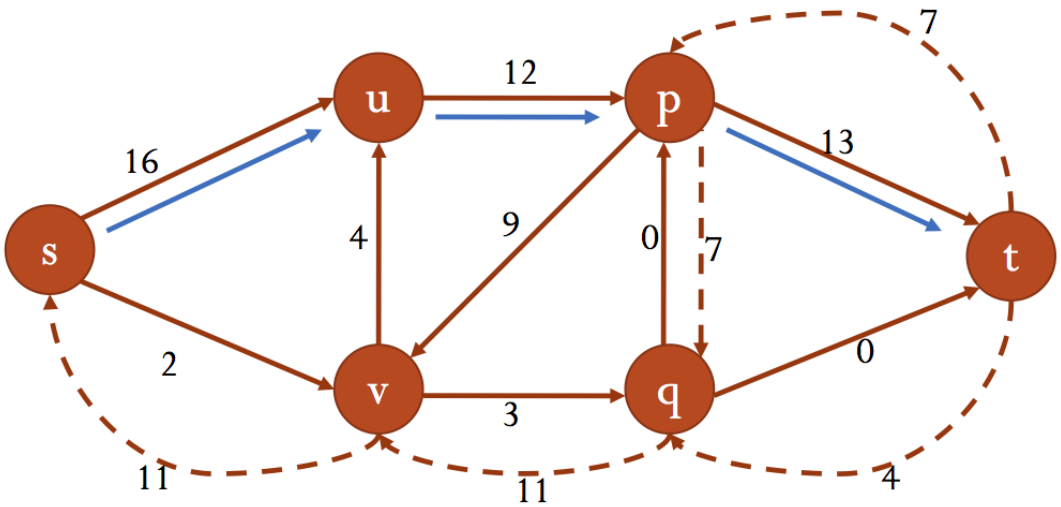



Figure: Graph $G_f$, where $f(s,u) = 0, f(s,v) = 11, f(v,u) = 0, f(v,q) = 11, f(u,p) = 0, f(p,v) = 0, f(p,t) = 7, f(q,p) = 7, f(q,t) = 4$

## Algorithm

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)



**Algorithm**

`Ford-Fulkerson`

- Start with a flow $f$ such that $f(e) = 0$
- While there is an $s - t$ path $P$ in $G_f$
  - Augment flow along an $s - t$ path and let $f'$ be resulting flow
  - Update $f$ to $f'$ and $G_f$ to $G_{f'}$
- return($f$)

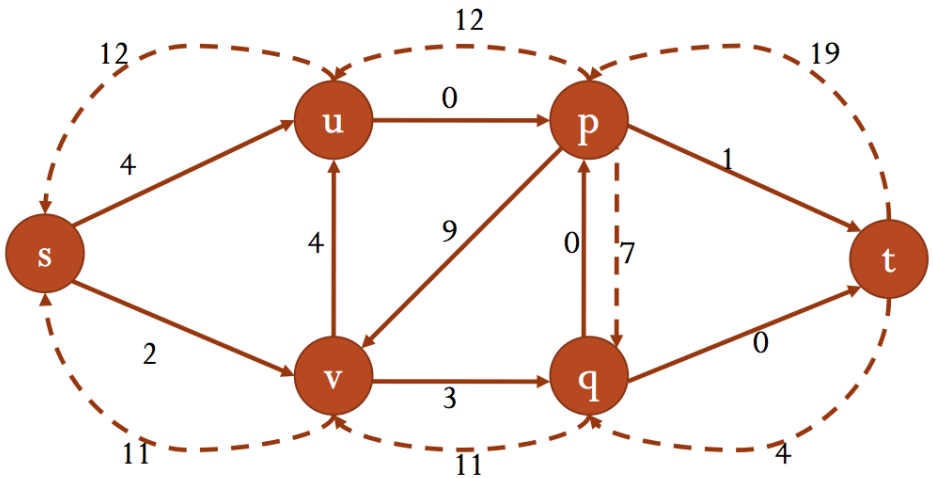



Figure: Graph $G_f$, where $f(s,u) = 12, f(s,v) = 11, f(v,u) = 0, f(v,q) = 11, f(u,p) = 12, f(p,v) = 0, f(p,t) = 19, f(q,p) = 7, f(q,t) = 4$

End