CSL 356: Analysis and Design of Algorithms

Ragesh Jaiswal

CSE, IIT Delhi

Network Flow

Ford-Fulkerson algorithm

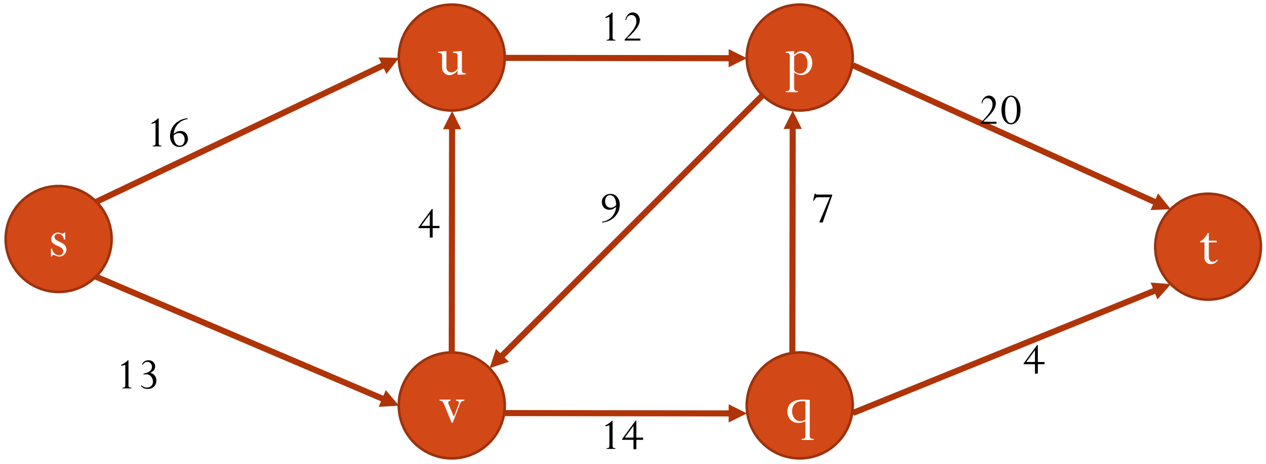
Network Flow

Max-Flow *//Ford-Fulkerson algorithm*

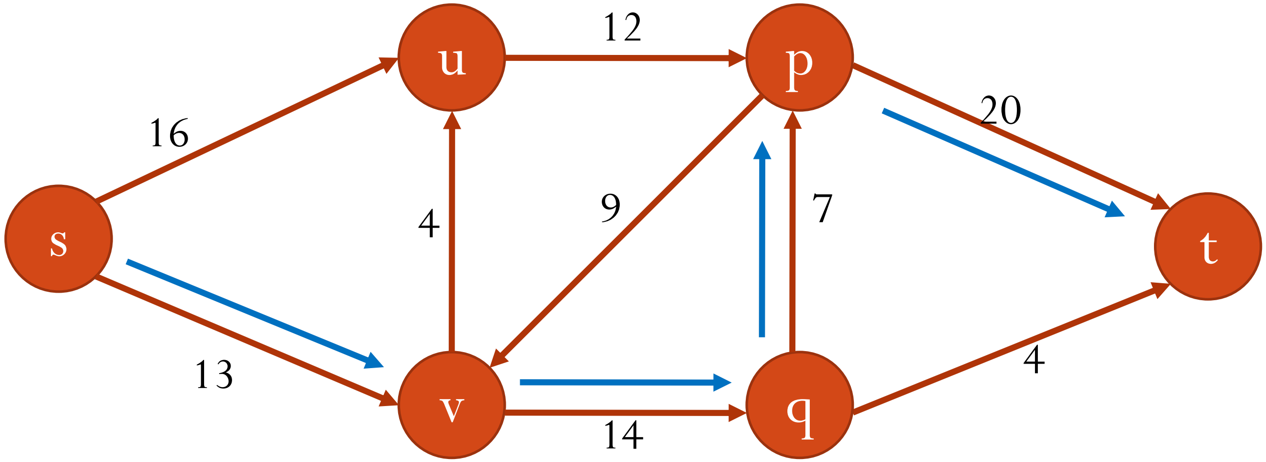
- Start with a flow f such that $f(e) = 0$
- while there is an $s - t$ path P in G_f
    - Execute the augmenting path algorithm to obtain f'
    - Update f to f' and G_f to $G_{f'}$
- return f

- Running time: $O(m \cdot C)$

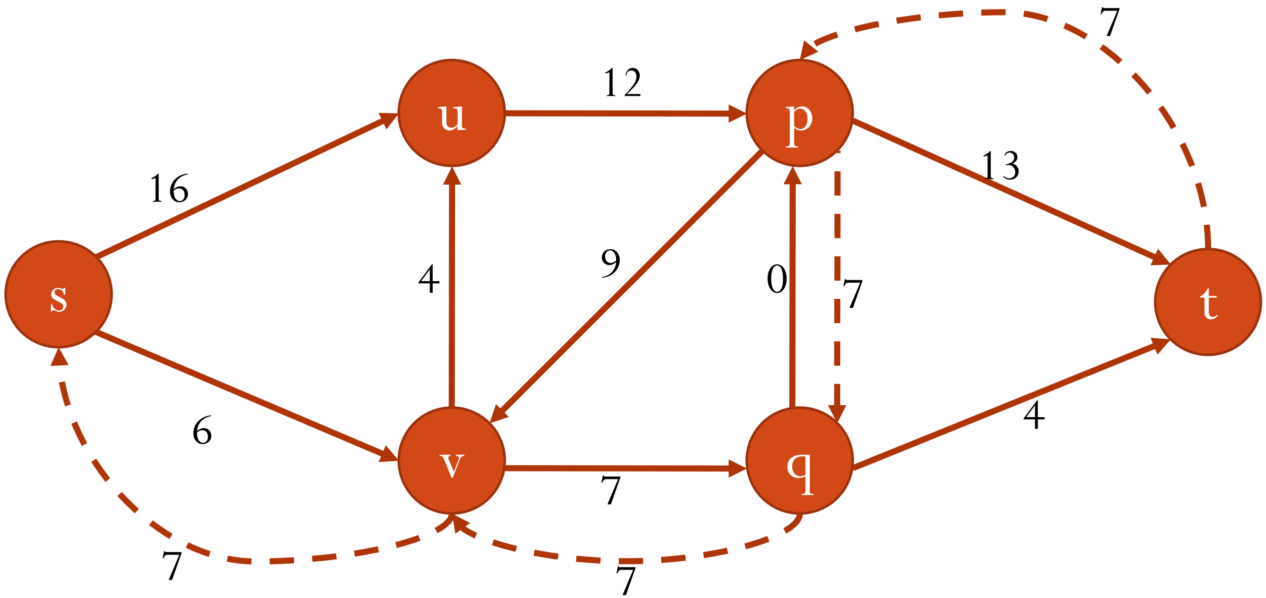
Network Flow



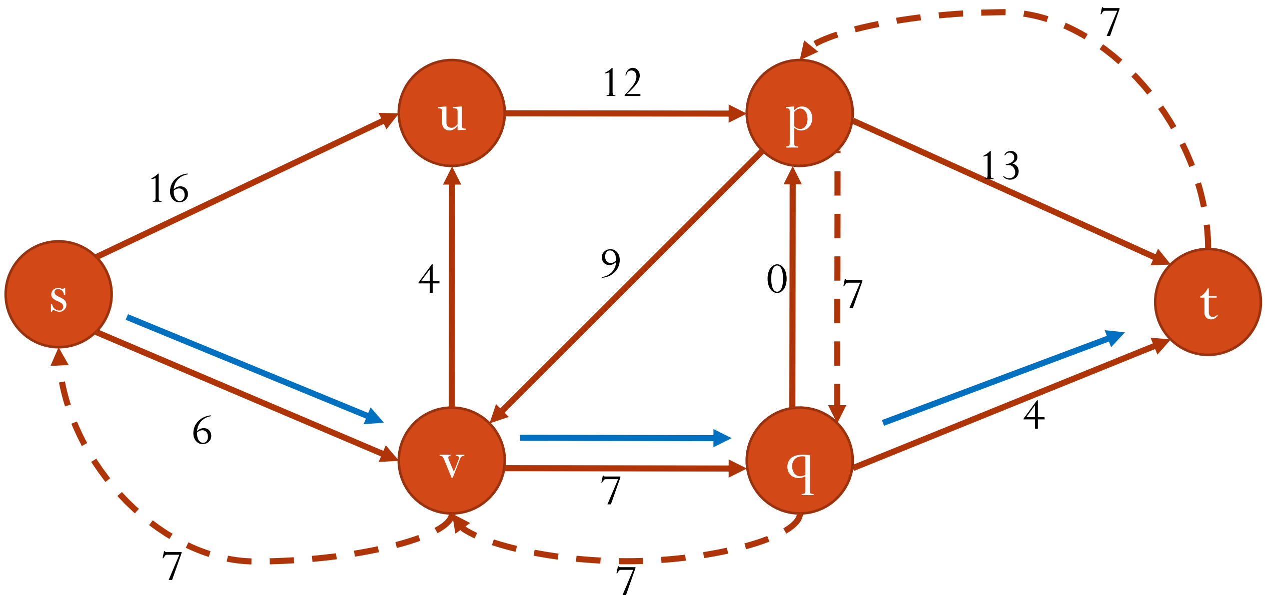
Network Flow



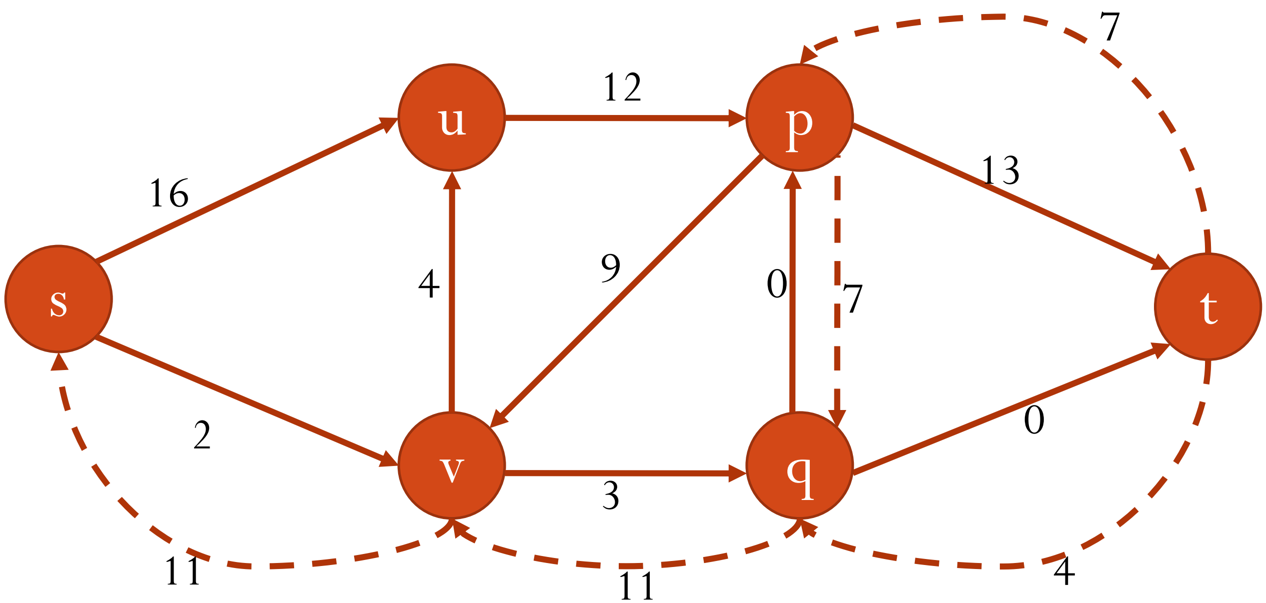
Network Flow



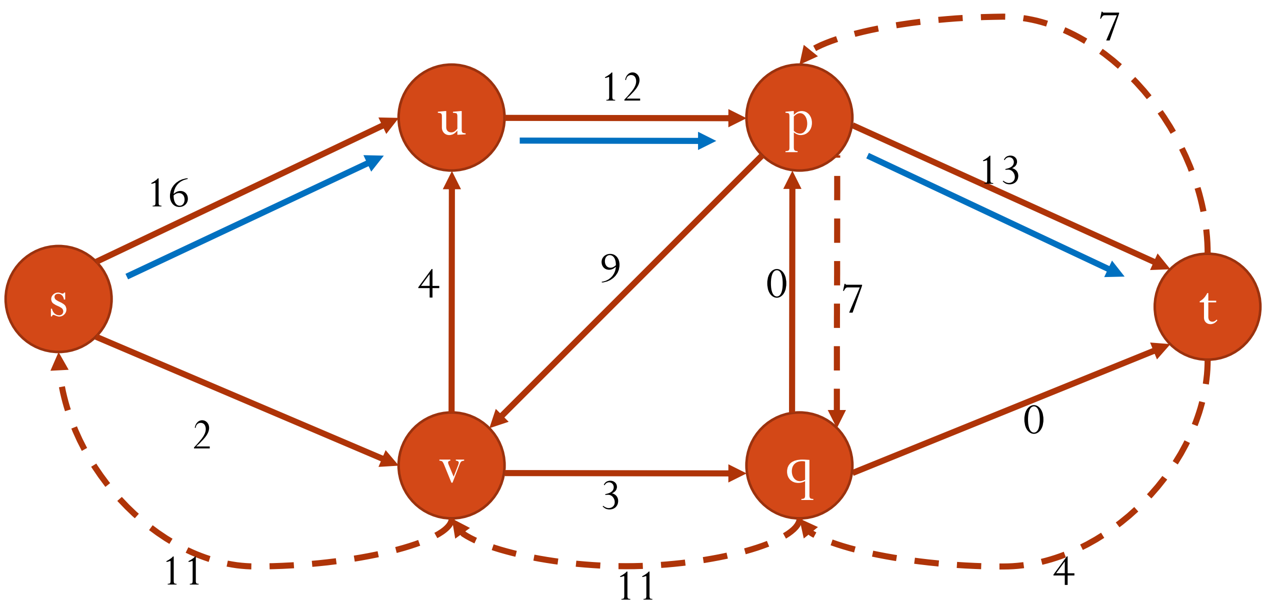
Network Flow



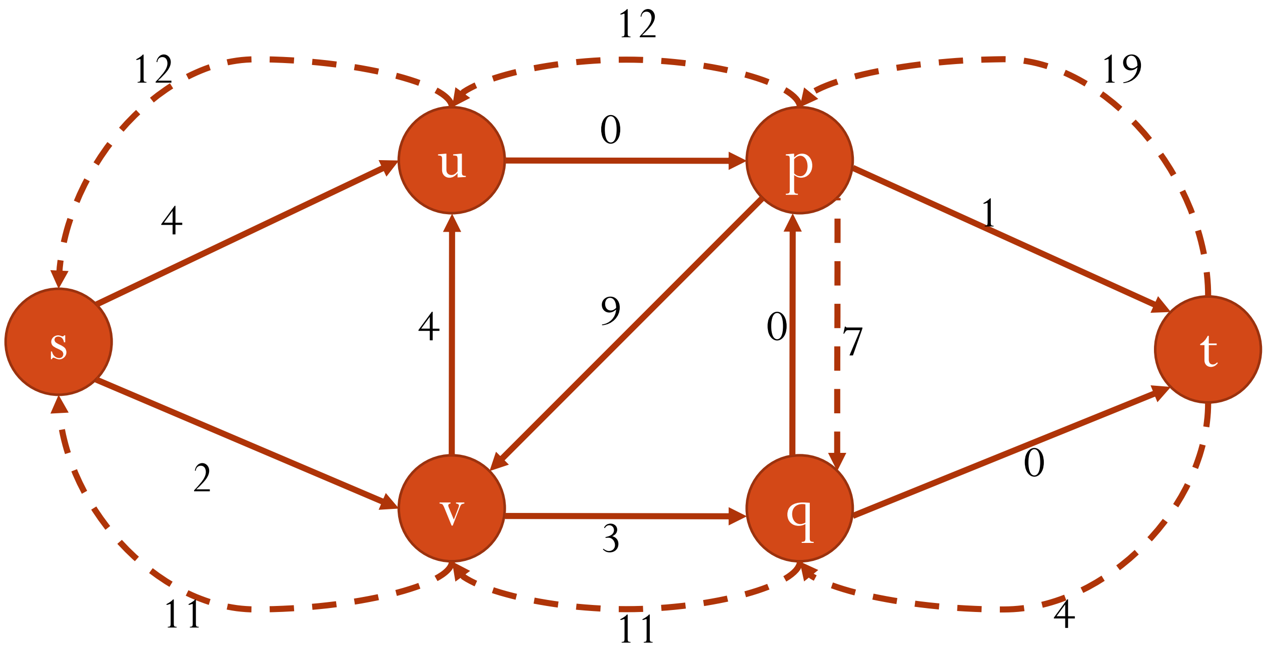
Network Flow



Network Flow



Network Flow



Network Flow

Ford-Fulkerson algorithm: Proof of correctness

Network Flow

- Theorem: Let f be the flow returned by the Ford-Fulkerson algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

- Let S be a subset of vertices and f be a flow. Then $f^{in}(S) = \sum_{e \text{ into } S} f(e)$ and $f^{out}(S) = \sum_{e \text{ out of } S} f(e)$

- *s-t cut*: A partition of vertices (A, B) is called an $s - t$ cut if A contains s and B contains t.

- *Capacity of s-t cut*: The capacity of an $s - t$ cut (A, B) is defined as $C(A, B) = \sum_{e \text{ out of } A} c(e)$.

Network Flow

- Theorem: Let f be the flow returned by the Ford-Fulkerson algorithm. Then f maximizes $v(f) = \sum_{e\ out\ of\ s} f(e)$.

- Claim 1: For any $s - t$ cut (A, B) and any flow f,

$$v(f) = f^{out}(A) - f^{in}(A)$$

Network Flow

- Theorem: Let f be the flow returned by the Ford-Fulkerson algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

- Claim 1: For any $s - t$ cut (A, B) and any flow f,

$$v(f) = f^{out}(A) - f^{in}(A)$$

- Proof: $v(f) = f^{out}(\{s\}) - f^{in}(\{s\})$ and for all other nodes v in A we have $f^{out}(\{v\}) - f^{in}(\{v\}) = 0$. So,
  - $v(f) = \sum_{v \text{ in } A}(f^{out}(\{v\}) - f^{in}(\{v\})) = f^{out}(A) - f^{in}(A)$

Network Flow

- Theorem: Let f be the flow returned by the Ford-Fulkerson algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

- Claim 1: For any $s - t$ cut (A, B) and any flow f,

$$v(f) = f^{out}(A) - f^{in}(A)$$

- Proof: $v(f) = f^{out}(\{s\}) - f^{in}(\{s\})$ and for all other nodes v in A we have $f^{out}(\{v\}) - f^{in}(\{v\}) = 0$. So,
  - $v(f) = \sum_{v \text{ in } A}(f^{out}(\{v\}) - f^{in}(\{v\})) = f^{out}(A) - f^{in}(A)$

- Claim 2: Let f be any $s - t$ flow and (A, B) be any $s - t$ cut. Then $v(f) \le C(A, B)$.

Network Flow

- Theorem: Let f be the flow returned by the Ford-Fulkerson algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

- Claim 1: For any $s - t$ cut (A, B) and any flow f,
$$v(f) = f^{out}(A) - f^{in}(A)$$
- Proof: $v(f) = f^{out}(\{s\}) - f^{in}(\{s\})$ and for all other nodes v in A we have $f^{out}(\{v\}) - f^{in}(\{v\}) = 0$. So,
  - $v(f) = \sum_{v \text{ in } A}(f^{out}(\{v\}) - f^{in}(\{v\})) = f^{out}(A) - f^{in}(A)$

- Claim 2: Let f be any $s - t$ flow and (A, B) be any $s - t$ cut. Then $v(f) \le C(A, B)$.
- Proof: $v(f) = f^{out}(A) - f^{in}(A) \le f^{out}(A) \le C(A, B)$.

Network Flow

- Theorem: Let f be the flow returned by the Ford-Fulkerson algorithm. Then f maximizes $v(f) = \sum_{e \; out \; of \; s} f(e)$.

- Claim 3: Let f be a flow such that there is no $s - t$ path in G_f. Then there is an $s - t$ cut (A^*, B^*) such that $v(f) = C(A^*, B^*)$. Furthermore, f is a flow with maximum value and (A^*, B^*) is the $s - t$ cut with minimum capacity.

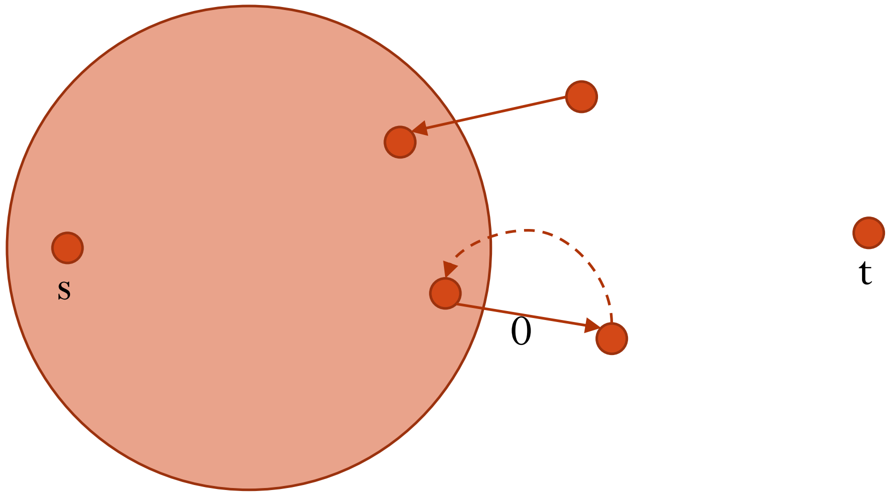
Network Flow

- <u>Theorem</u>: Let f be the flow returned by the Ford-Fulkerson algorithm. Then f maximizes $v(f) = \sum_{e \text{ out of } s} f(e)$.

- <u>Claim 3</u>: Let f be a flow such that there is no $s-t$ path in G_f. Then there is an $s-t$ cut (A^*, B^*) such that $v(f) = C(A^*, B^*)$. Furthermore, f is a flow with maximum value and (A^*, B^*) is the $s-t$ cut with minimum capacity.

- <u>Proof</u>:

$$\begin{aligned} v(f) &= f^{out}(A^*) - f^{in}(A^*) \\ &= f^{out}(A^*) - 0 \\ &= C(A^*, B^*) \end{aligned}$$



A^* (all vertices reachable from s in G_f)

Network Flow

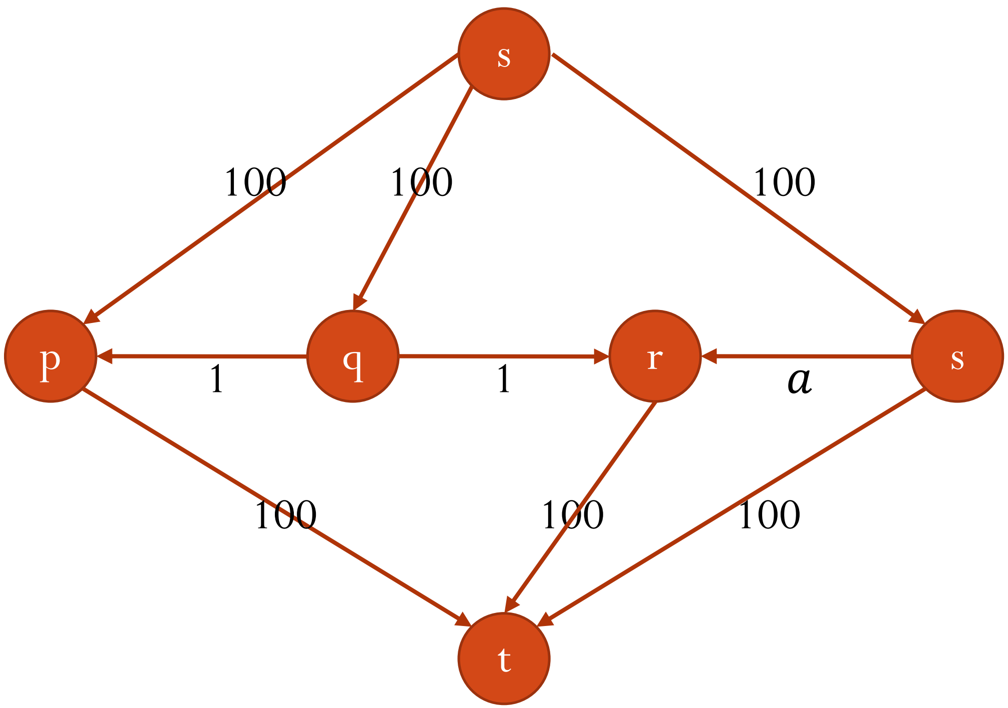
- Theorem(Max-flow-min-cut): In every flow network, the maximum value of an $s-t$ flow is equal to the minimum capacity of an $s-t$ cut.

Network Flow:

- Ford-Fulkerson algorithm:
  - Given network with integer capacities, find a source-to-sink path and push as much flow along the path as possible.
  - Update the residual capacity of edges in the residual graph.
  - Repeat.
- Proof of correctness:
  - The algorithm terminates.
    - The capacities are integers.
    - What if the capacities are not integers? Does the algorithm terminate?
  - Max-flow-min-cut theorem: In every network flow the maximum value of an $s - t$ flow is equal to the minimum capacity of an $s - t$ cut.

Network Flow:

- A simple example where the Ford-Fulkerson algorithm does not terminate.

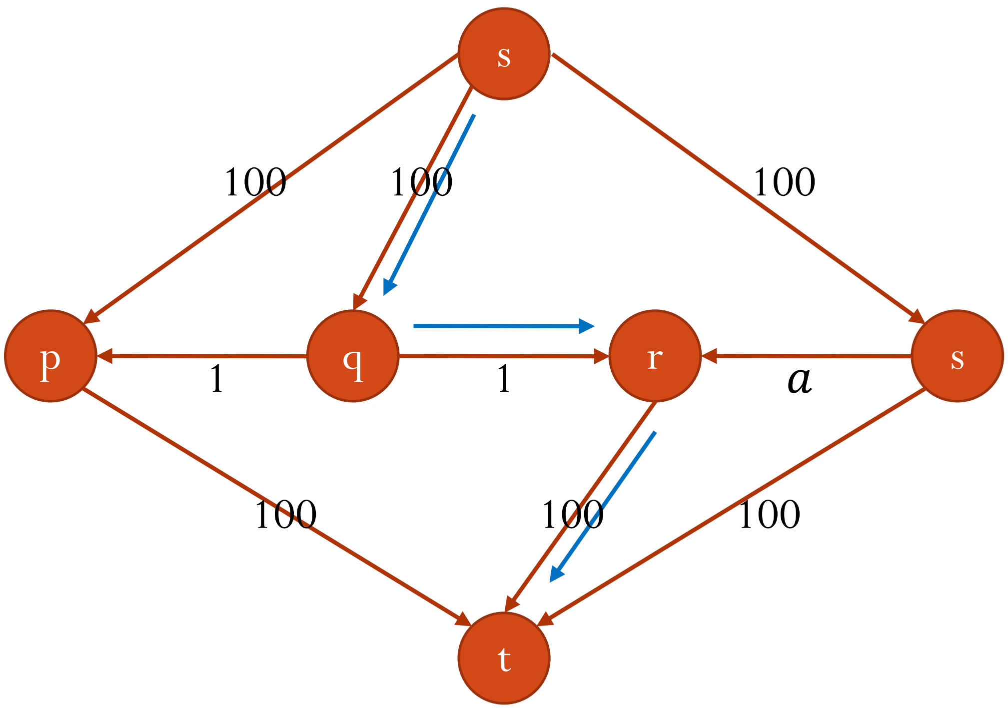


$$1 - a = a^2$$

Network Flow:

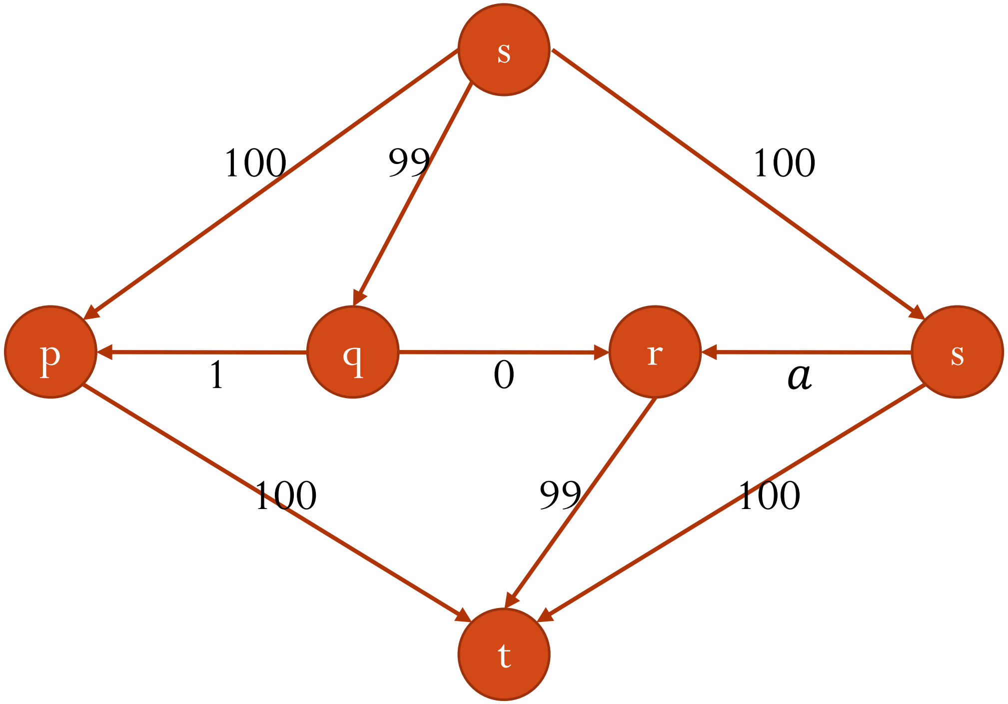
- A simple example where the Ford-Fulkerson algorithm does not terminate.

$$1 - a = a^2$$



Network Flow:

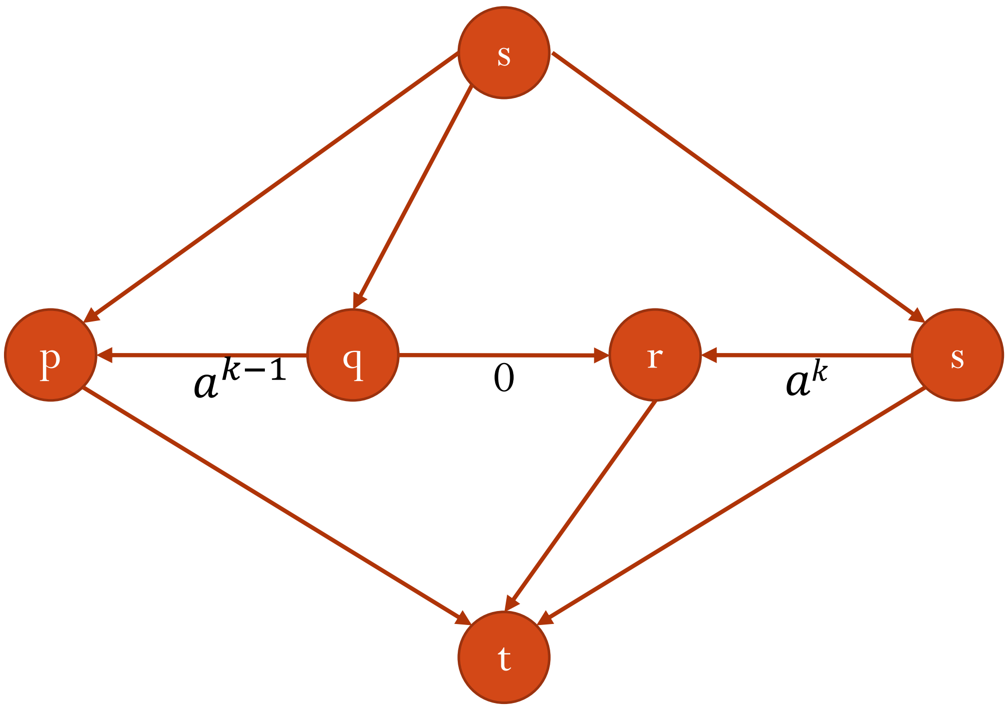
- A simple example where the Ford-Fulkerson algorithm does not terminate.



$$1 - a = a^2$$

Network Flow:

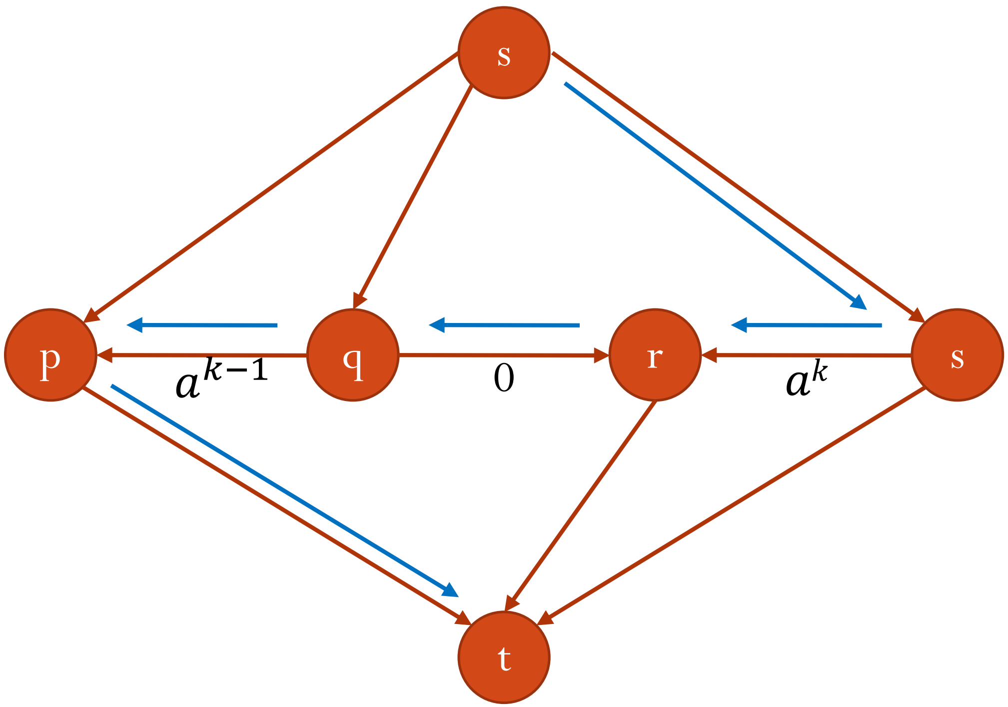
- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q, p), (q, r), and (s, r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

Network Flow:

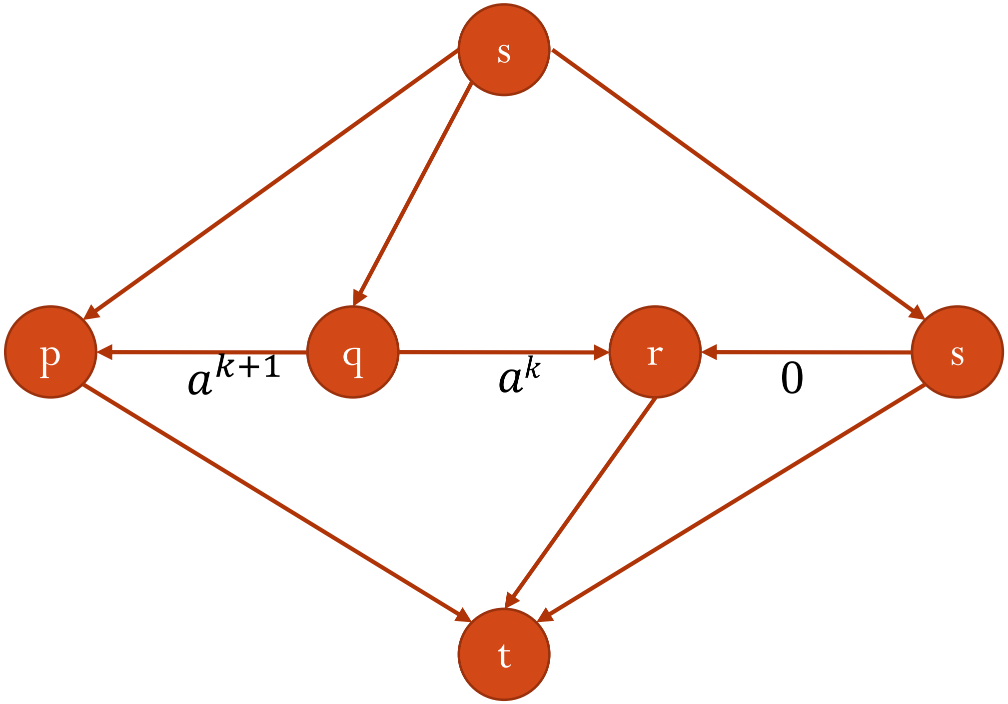
- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q,p), (q,r), and (s,r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

Network Flow:

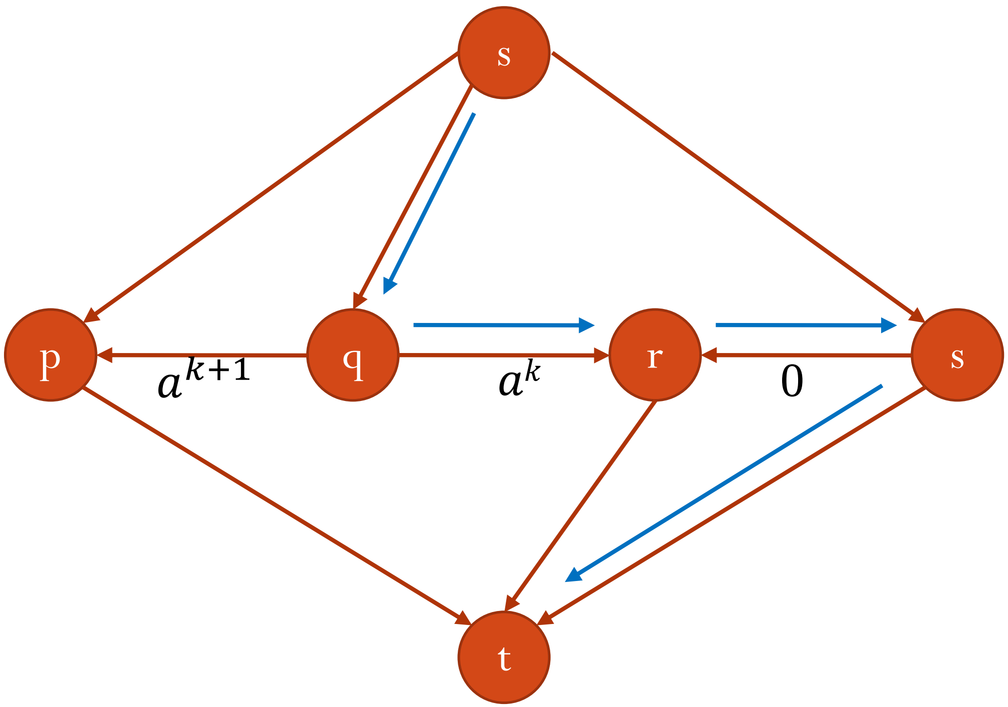
- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q, p), (q, r), and (s, r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

Network Flow:

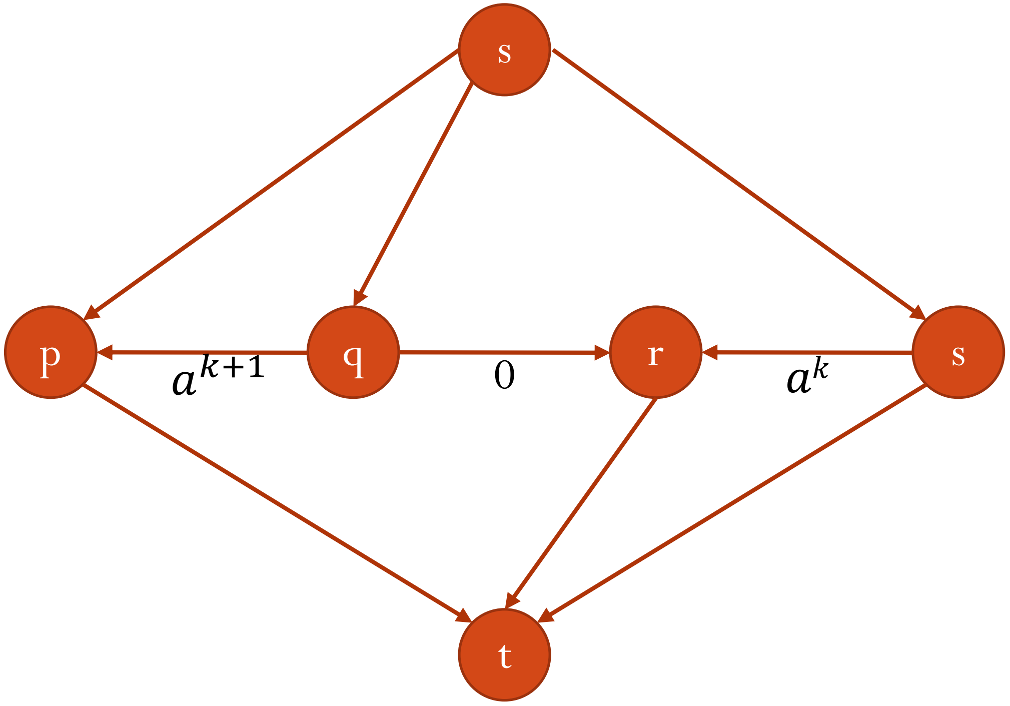
- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q,p), (q,r), and (s,r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

Network Flow:

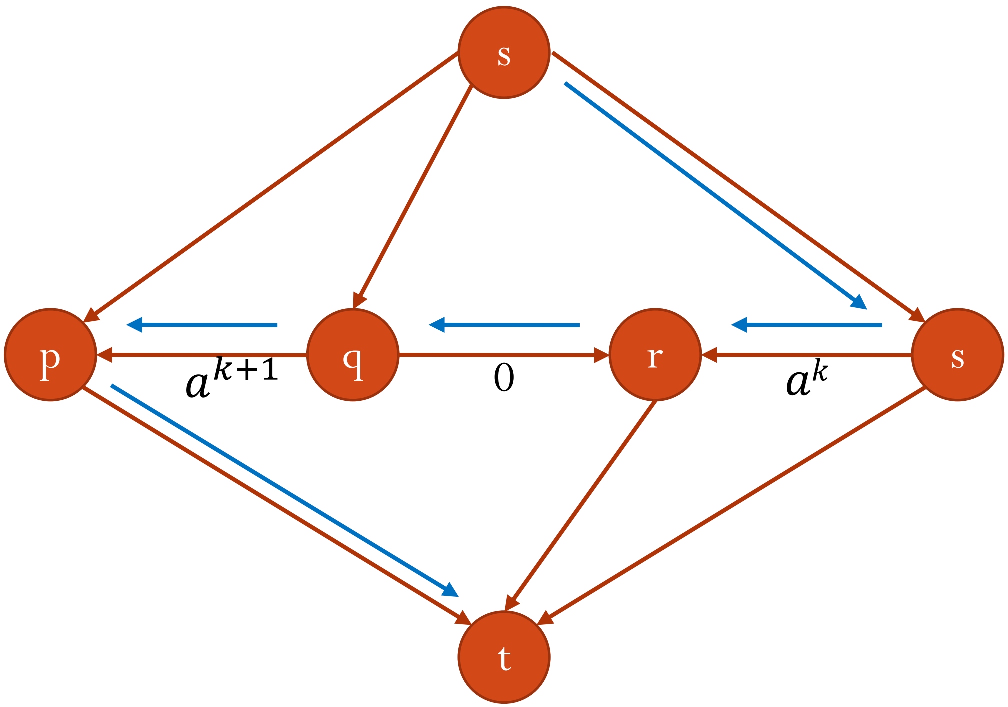
- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q, p), (q, r), and (s, r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

Network Flow:

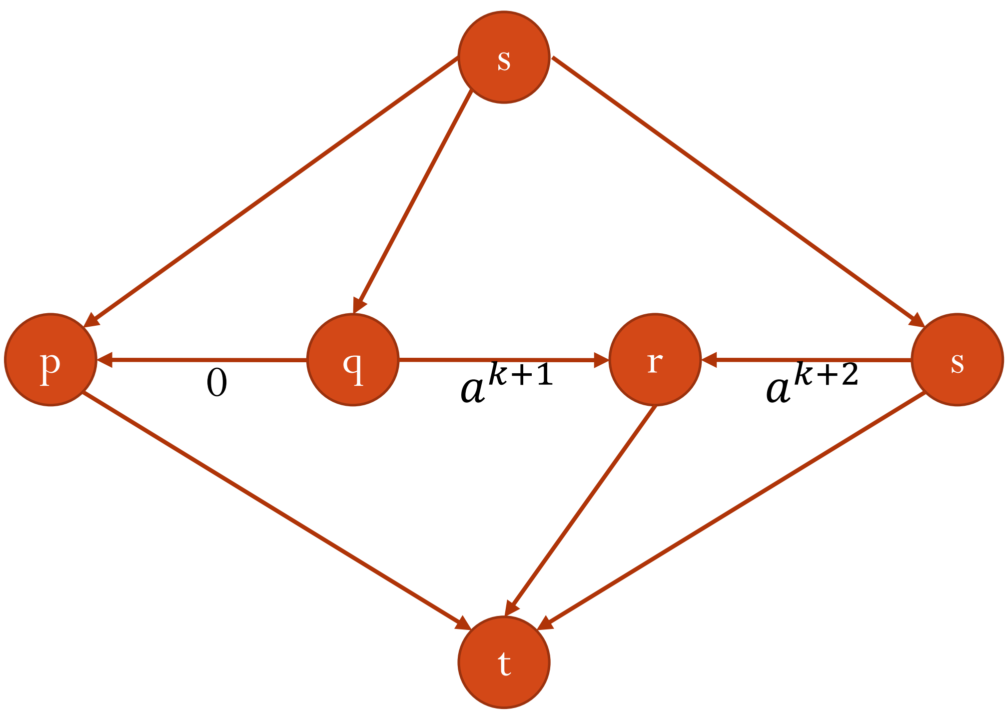
- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q, p), (q, r), and (s, r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

Network Flow:

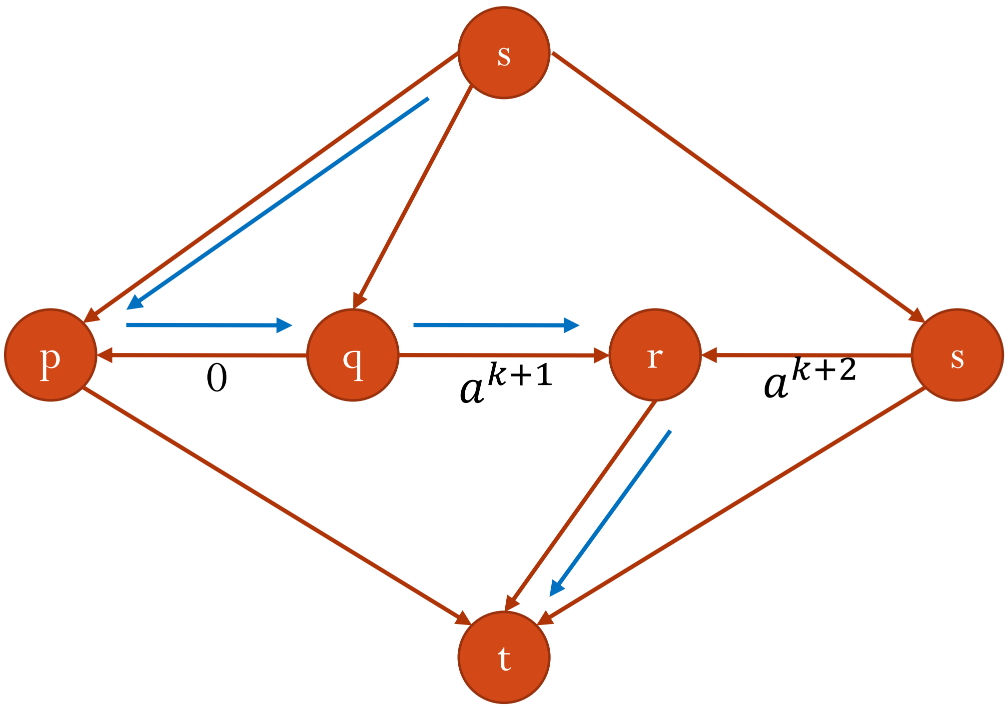
- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q, p), (q, r), and (s, r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

Network Flow:

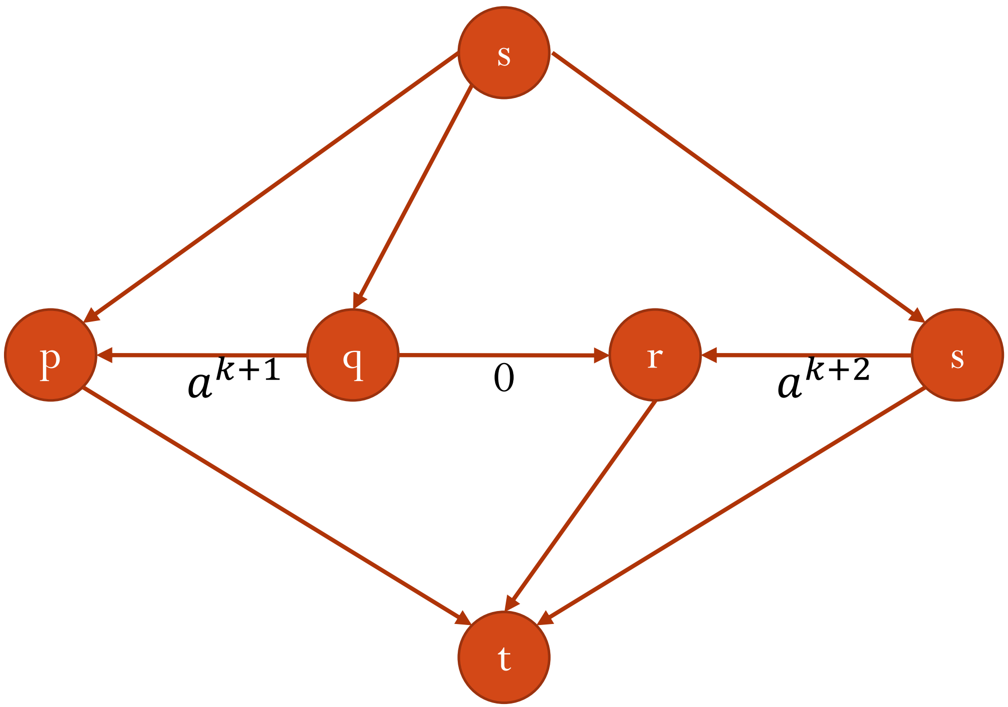
- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q, p), (q, r), and (s, r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

Network Flow:

- A simple example where the Ford-Fulkerson algorithm does not terminate.
- Suppose inductively, the residual capacities of edges (q,p), (q,r), and (s,r) are $a^{k-1}, 0, a^k$. Consider next four flows.



$$1 - a = a^2$$

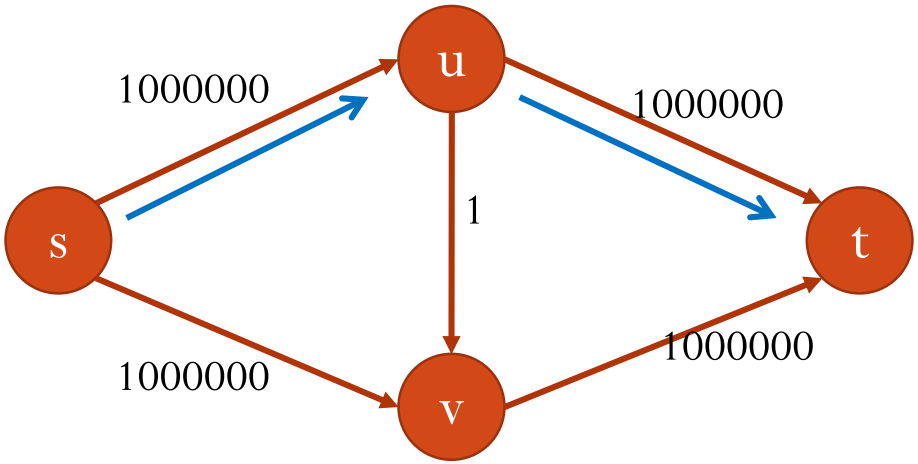
Network Flow:

- A simple example where the Ford-Fulkerson algorithm does not terminate.
- The total value of the flow converges to $(1 + 2\sum a^i) = 4 + \sqrt{5}$.
- The max flow is 201.

Network Flow: running time

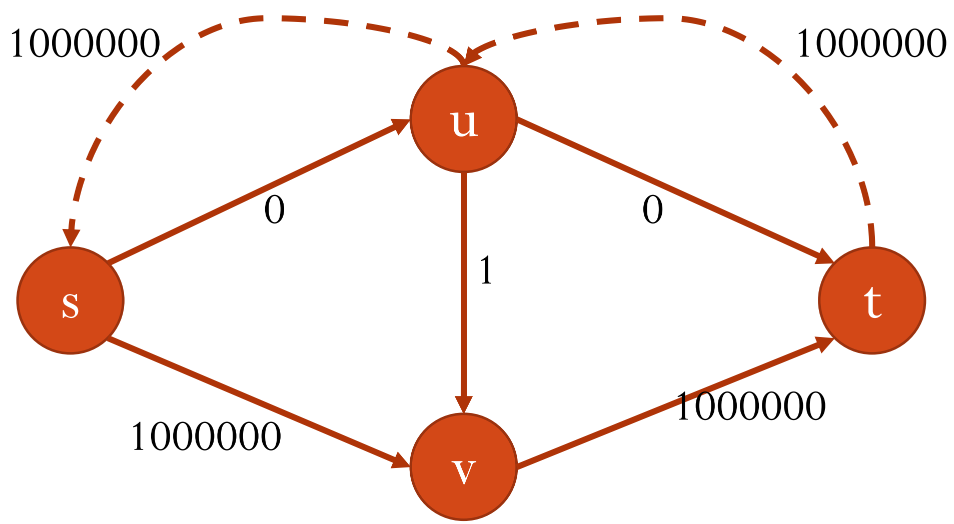
Network Flow

- $C = \sum_{e\, out\, of\, s} c(e)$
- The running time of the Ford-Fulkerson algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - Example:



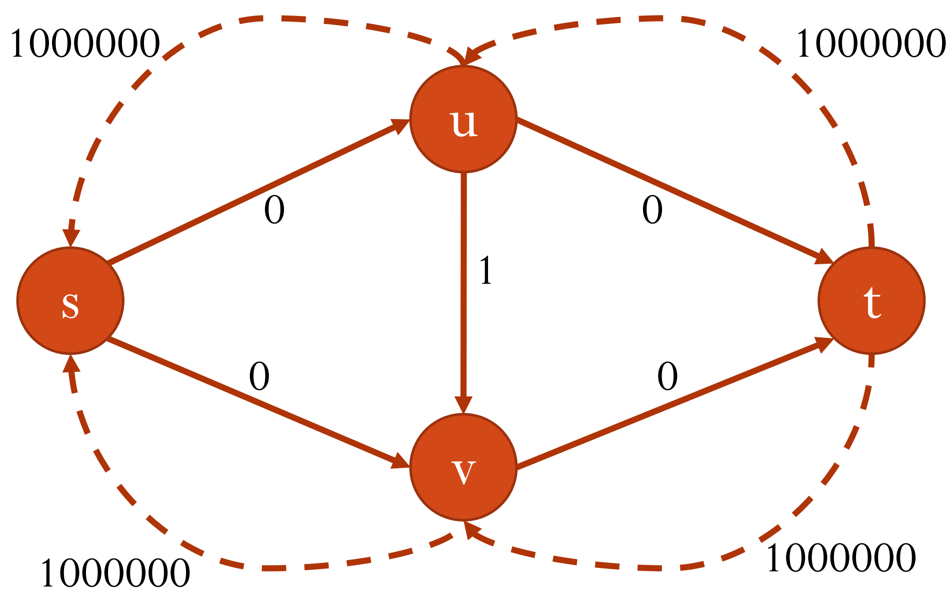
Network Flow

- $C = \sum_{e \, out of \, s} c(e)$
- The running time of the Ford-Fulkerson algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - Example:



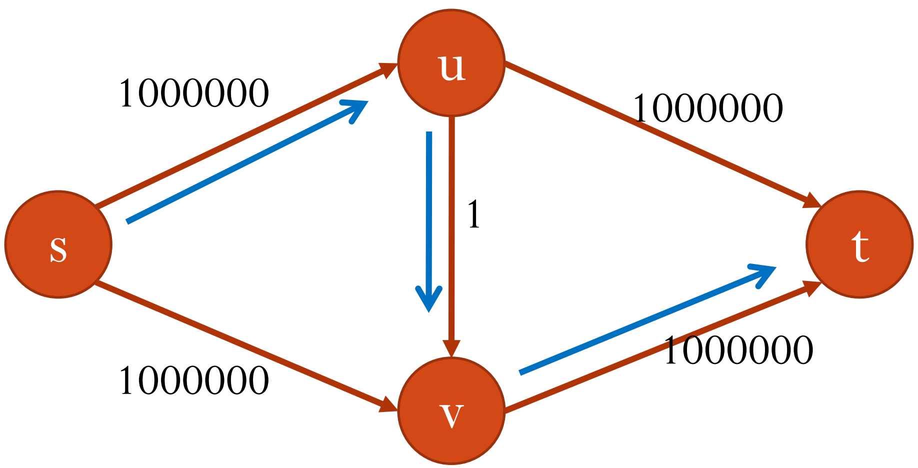
Network Flow

- $C = \sum_{e \, out of \, s} c(e)$
- The running time of the Ford-Fulkerson algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - Example:



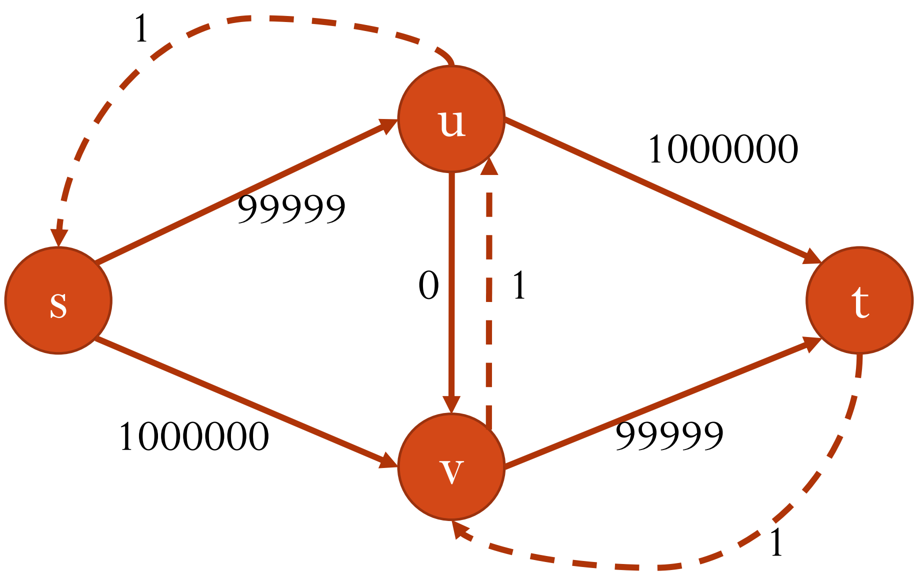
Network Flow

- $C = \sum_{e\, out of\, s} c(e)$
- The running time of the Ford-Fulkerson algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - Example:



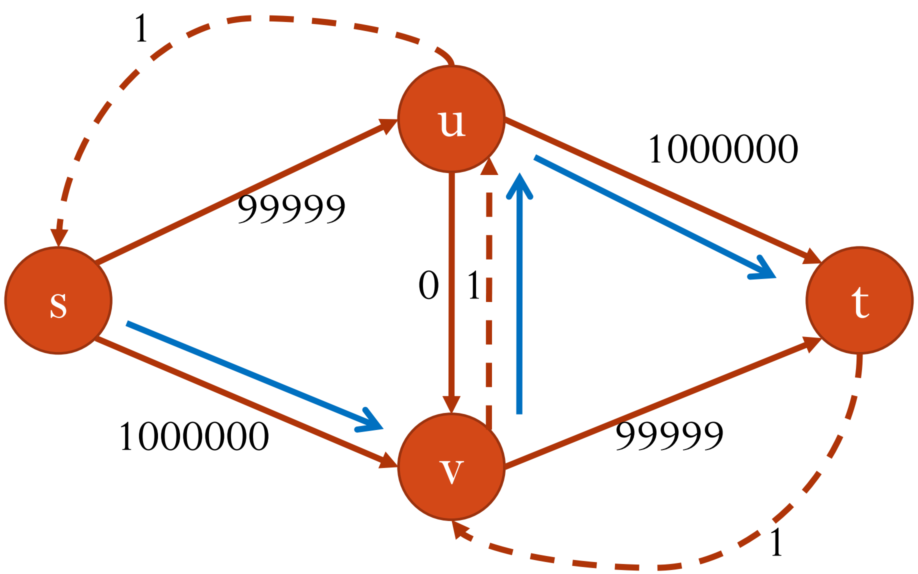
Network Flow

- $C = \sum_{e\, out of\, s} c(e)$
- The running time of the Ford-Fulkerson algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - Example:



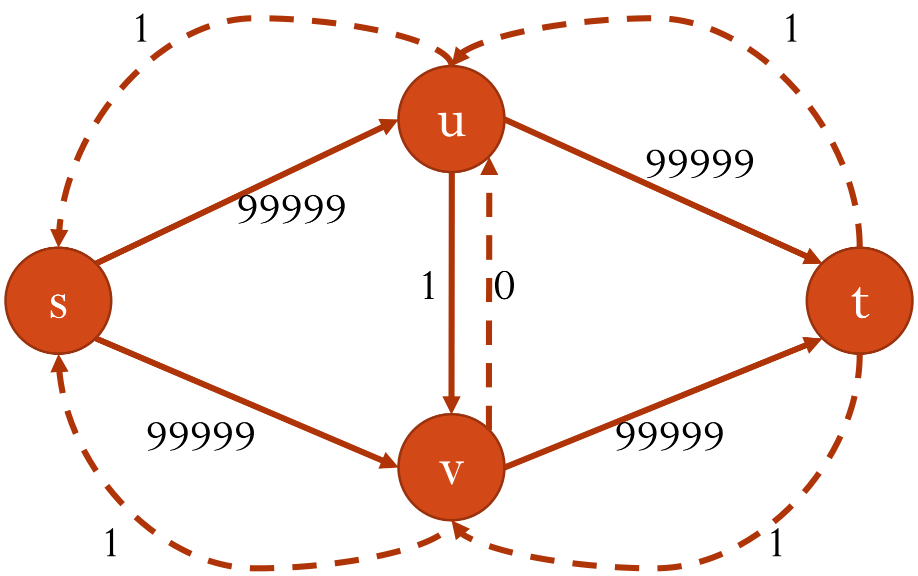
Network Flow

- $C = \sum_{e \, out of \, s} c(e)$
- The running time of the Ford-Fulkerson algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - Example:



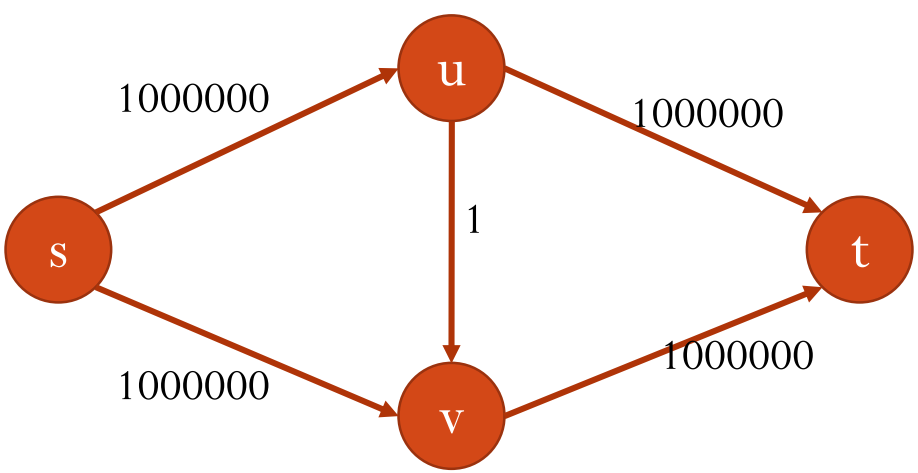
Network Flow

- $C = \sum_{e\, out of\, s} c(e)$
- The running time of the Ford-Fulkerson algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - Example:



Network Flow

- $C = \sum_{e \text{ out of } s} c(e)$
- The running time of the Ford-Fulkerson algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - Example: We might get a better running time if we could hide the edge with small capacity when looking for an augmenting path.
- General idea: Use all edges with large capacities before considering edges with smaller capacity.



Network Flow

- For an $s - t$ flow f and a positive integer Δ, let $G_f(\Delta)$ denote a subset of the residual graph G_f consisting only of edges with residual capacity of at least Δ.
- Idea: Instead of finding augmenting paths in G_f, we will find augmenting paths in $G_f(\Delta)$ for smaller and smaller values of Δ.

Scaling-Max-Flow

- Start with an $s - t$ flow such that for all e, $f(e) = 0$
- Δ=largest power of 2 smaller than C
- while $\Delta \geq 1$
  - while there is an $s - t$ path P in $G_f(\Delta)$
    - Execute the augmenting path algorithm to obtain f'
    - Update f to f' and $G_f(\Delta)$ to $G_{f'}(\Delta)$
  - $\Delta = \Delta/2$
- return f

End