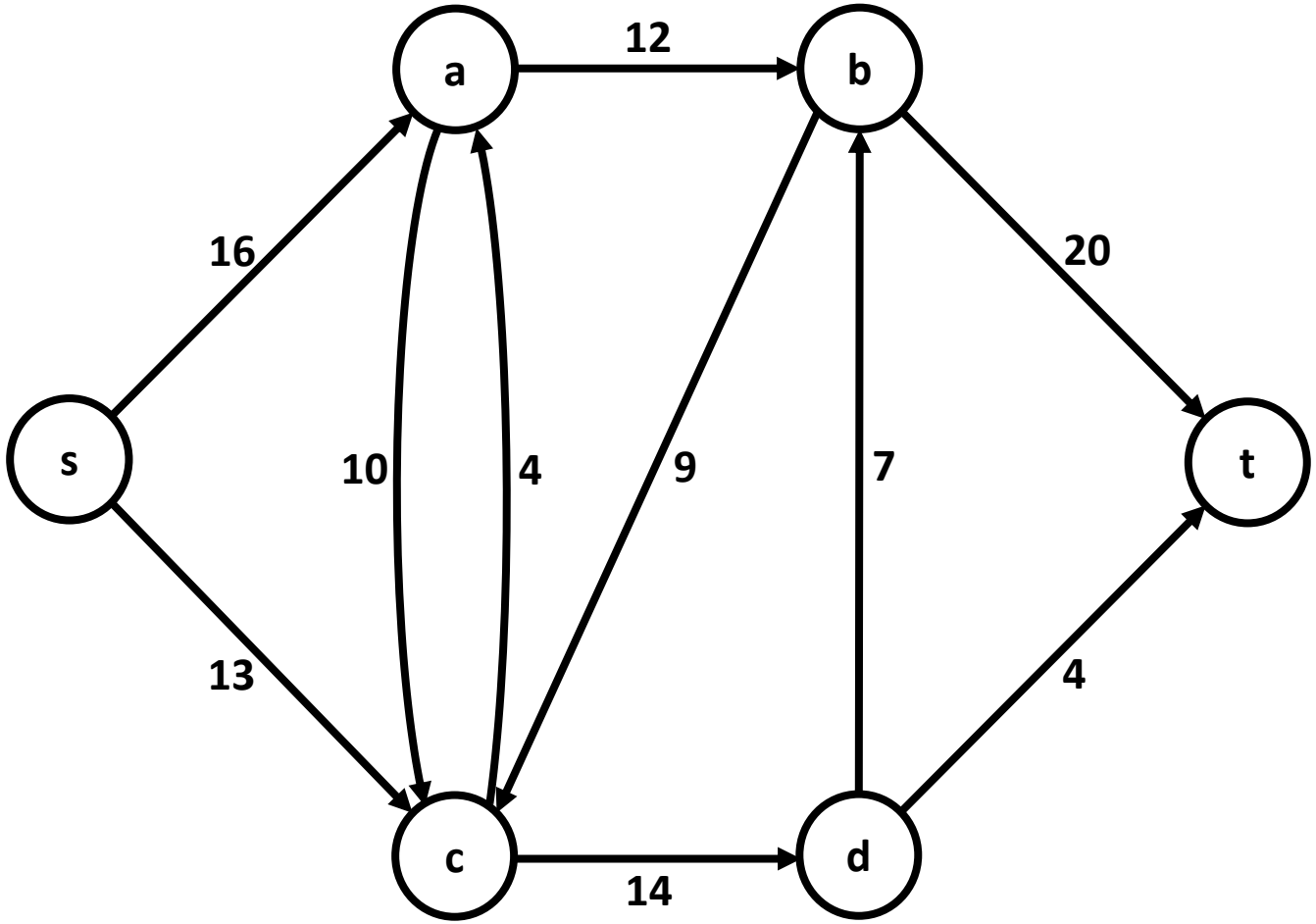
Flow Networks

Flow Network

- A flow network $G = (V, E)$ is a weighted directed graph
  - Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.
  - If (u, v) does not belong to E, $c(u, v) = 0$.
  - Two special vertices are considered: a source s and a sink t.

Maximum Flow Problem

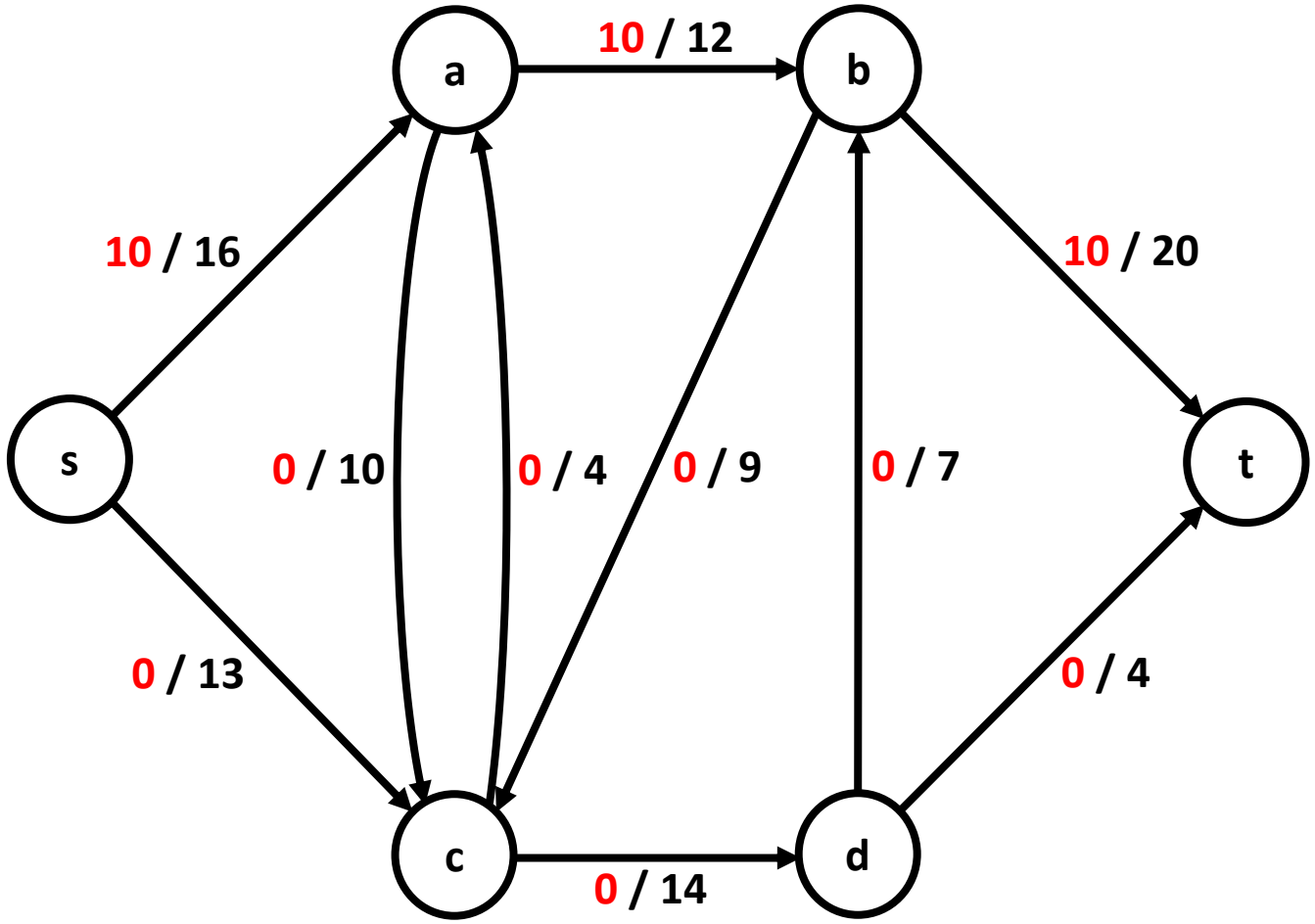
- Input: A directed graph, source vertex s, and sink vertex t. Each edge has a non-negative capacity.
- Assumption: No edge enters into s or leaves from t.



Maximum Flow Problem

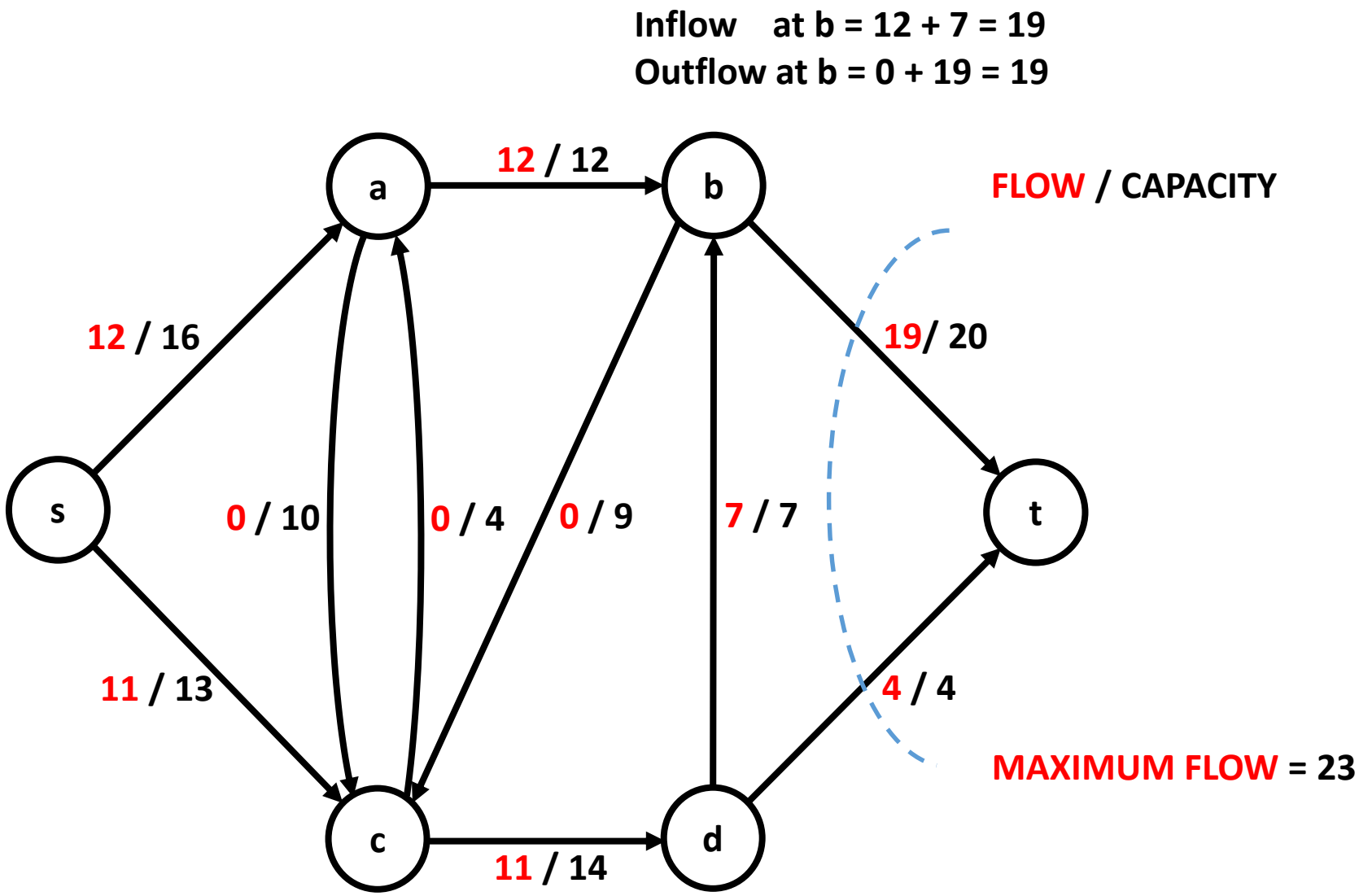
- A st-flow (flow) is an assignment of values to the edges such that:
  - Capacity constraint: 0 ≤ edge's flow ≤ edge's capacity.
  - Flow constraint: inflow = outflow at every vertex (except s and t).
- The value of a flow is the inflow at t (or outflow from s).
- Maximum st-flow (max flow) problem: Find a flow of maximum value.
- Output: Find a flow of maximum value.

Flow vs Capacity



FLOW / CAPACITY

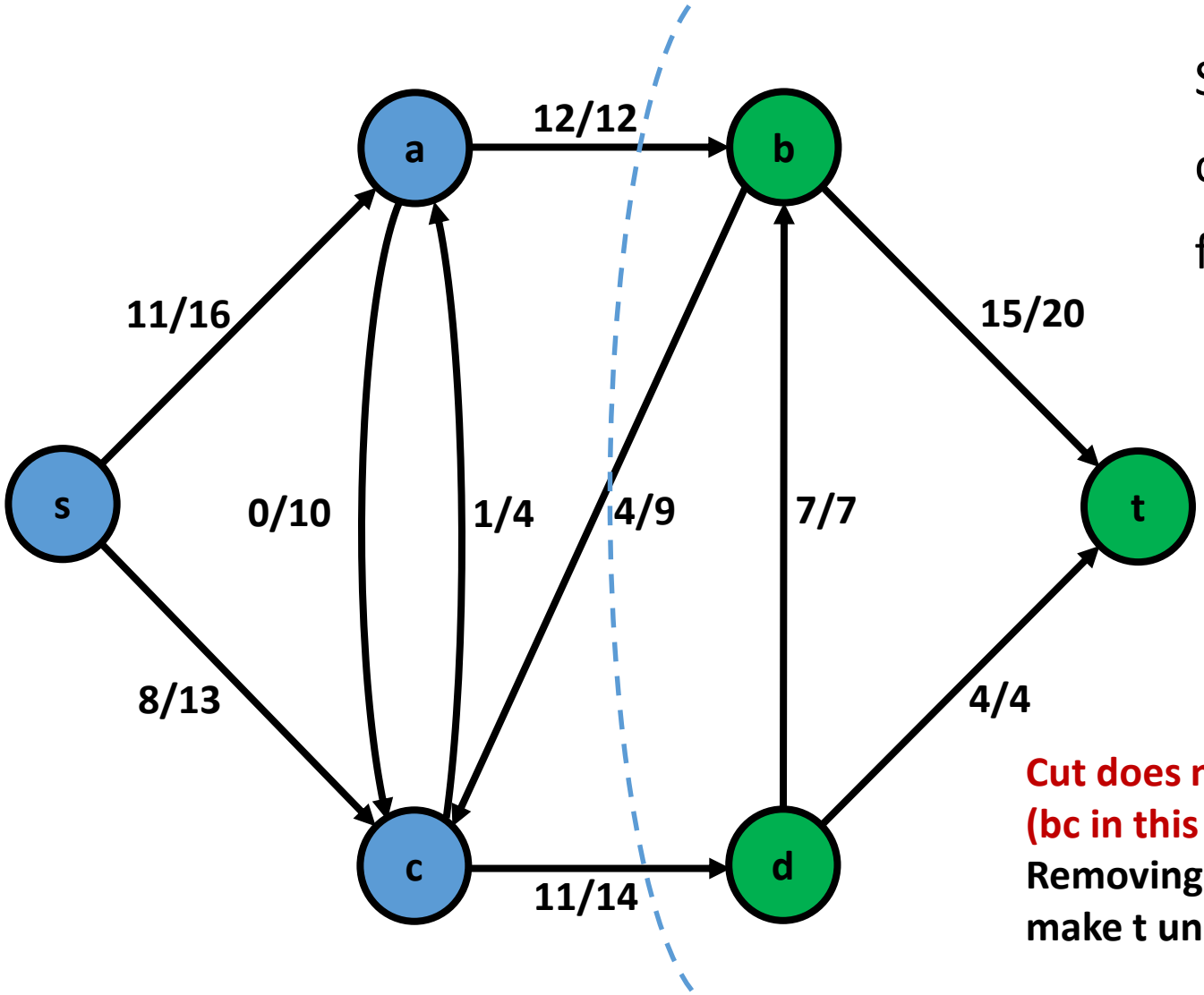
Maximum Flow



Minimum Cut Problem

- A st-cut is a partition of the vertices into two disjoint sets S and T with $s \in S$ and $t \in T$.
- Capacity of a st-cut c(S, T) is the sum of the capacities of the edges from S to T.
- Cut does not count edges from T to S - Why? Removing forward edges is enough to make t unreachable from s.
- If f is a flow, then the net flow across the cut (S, T) is defined to be f(S, T).
- Minimum st-cut (min cut) problem: Find a st-cut of minimum capacity.
- Output: Find a st-cut of minimum capacity.

Net Flow vs. Cut



S ={s, a, c} and T = {b, d, t}.

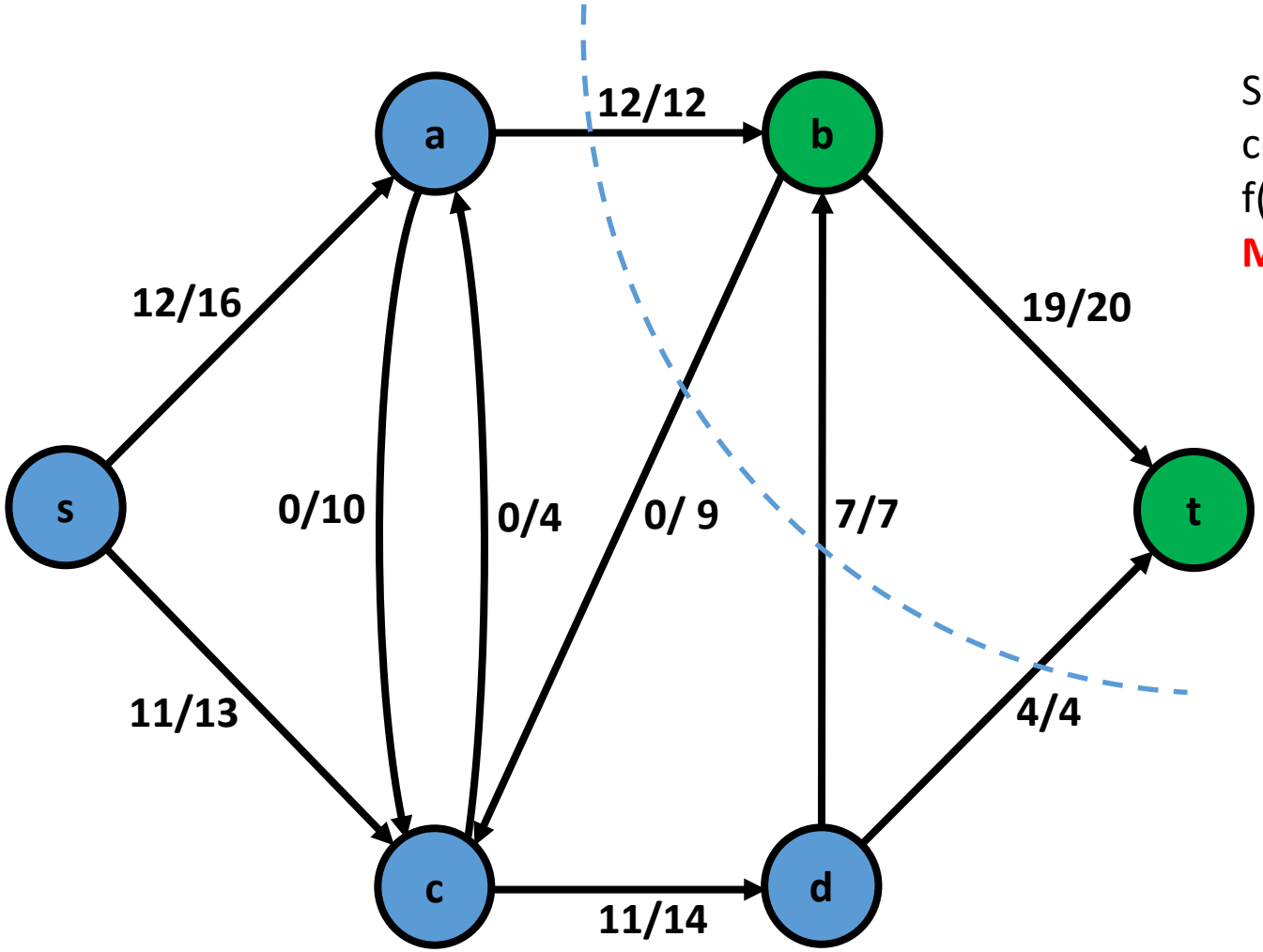
c(S, T) = 12 + 14 = 26.

f(S, T) = 12 – 4 + 11 = 19.

Cut does not count edges from T to S (bc in this example) Why?

Removing forward edges is enough to make t unreachable from s

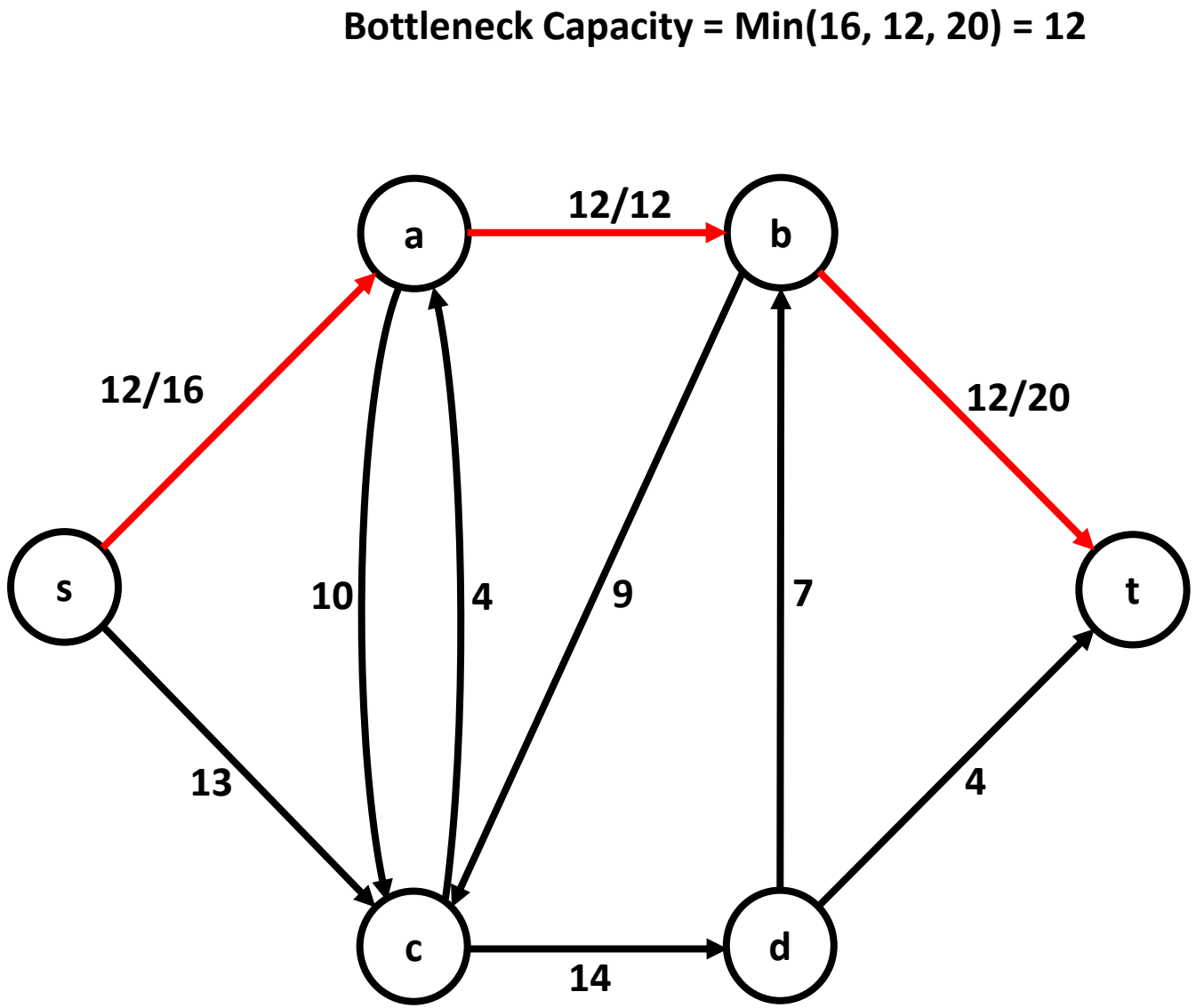
Minimum Cut



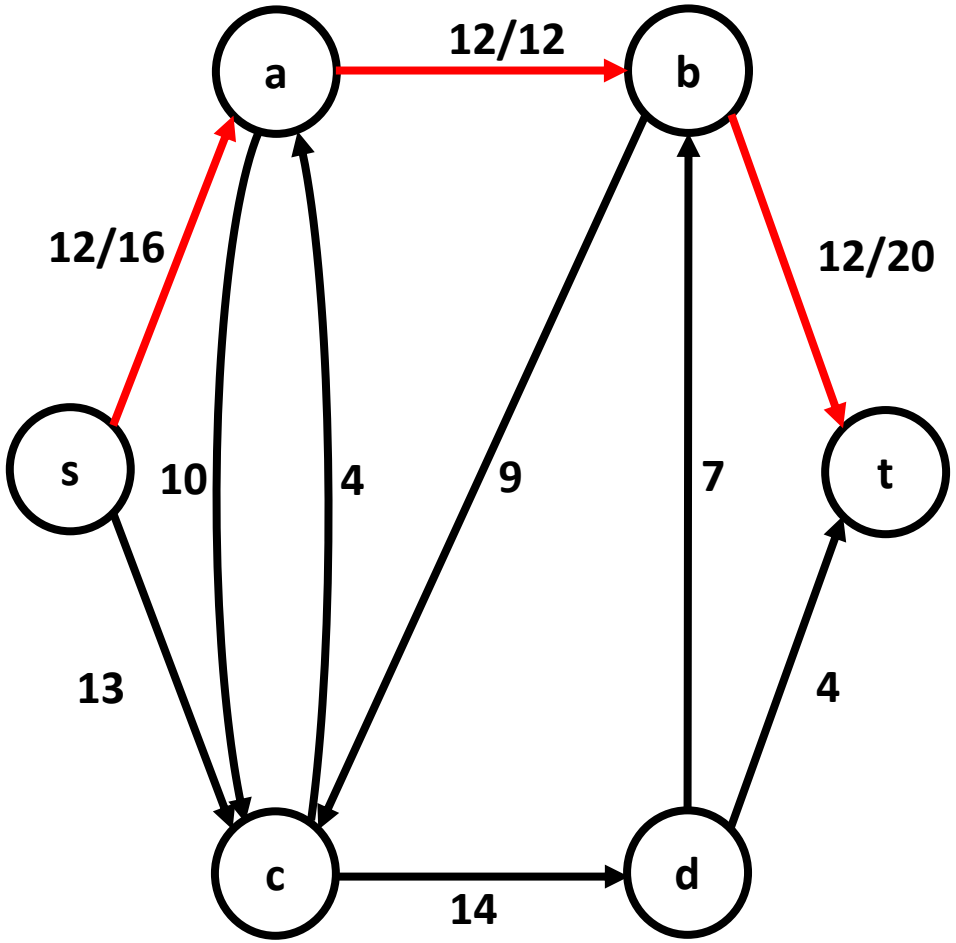
S ={s, a, c, d} and T = {b, t}.
c(S, T) = 12 + 7 + 4 = 23.
f(S, T) = 12 – 0 + 7 + 4 = 23.
MINIMUM CUT= 23

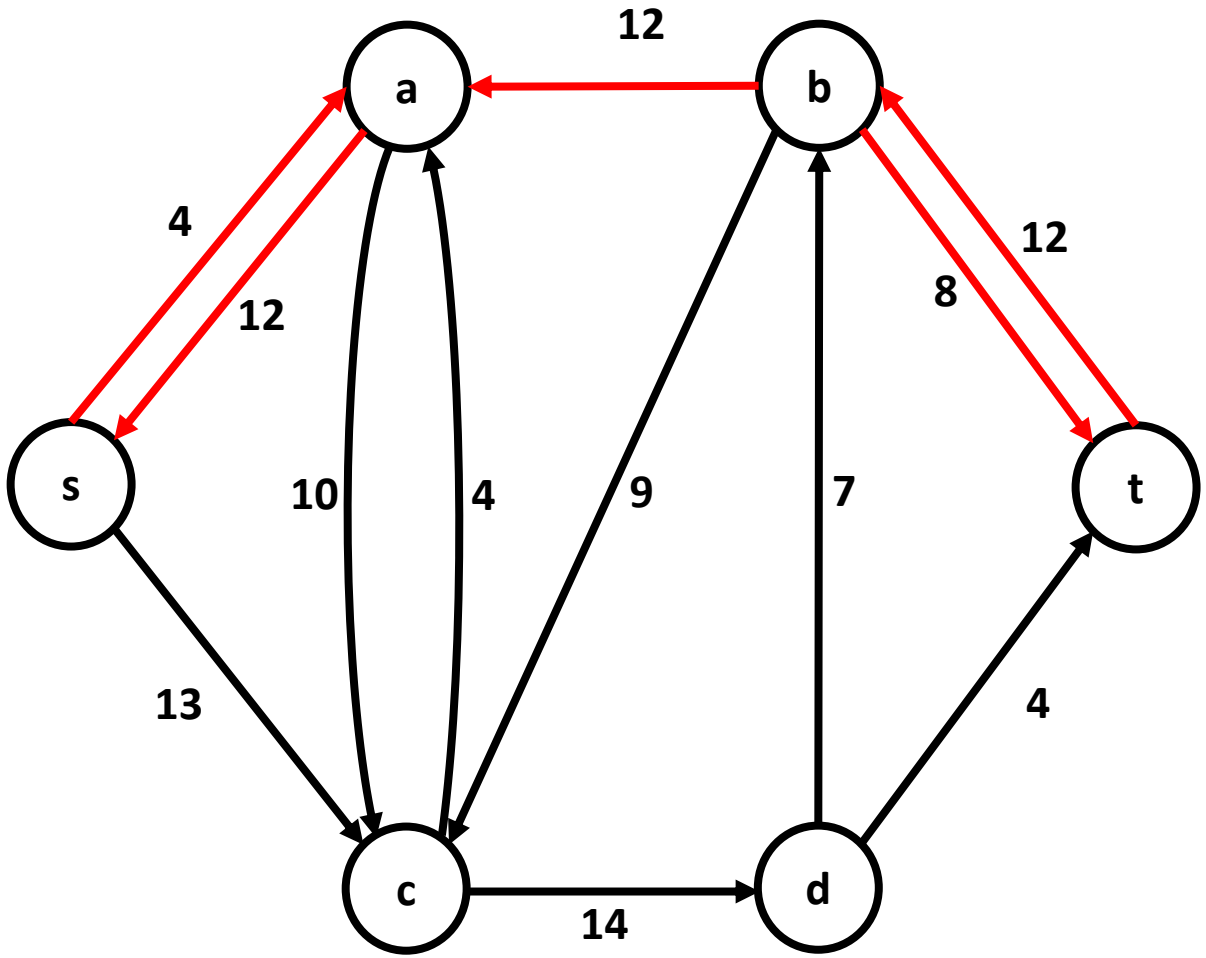
Few Definitions

- Augmenting Path: An s,t path with positive capacity at each edge along the path
- Bottleneck Capacity: The maximum amount of fluid that can be flown in an augmenting path.
- Residual Network: Given a flow network G=(V, E), the residual network is $G_R = (V, E_R)$ has edges (u, v) with weight c(u, v) – f(u, v)



Residual Network





Residual Network

- Given a flow network $G = (V, E)$ and a flow f , the **residual network** of G induced by f is $G_f = (V, E_f)$, where $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$.
- **Residual capacity** : $c_f(u, v) = c(u, v) – f(u, v)$
- Each edge of the residual network, or **residual edge**, can admit a flow $f > 0$.
- $|E_f| \leq 2|E|$.

Proof: The edges in E_f are either edges in E or their reversals.

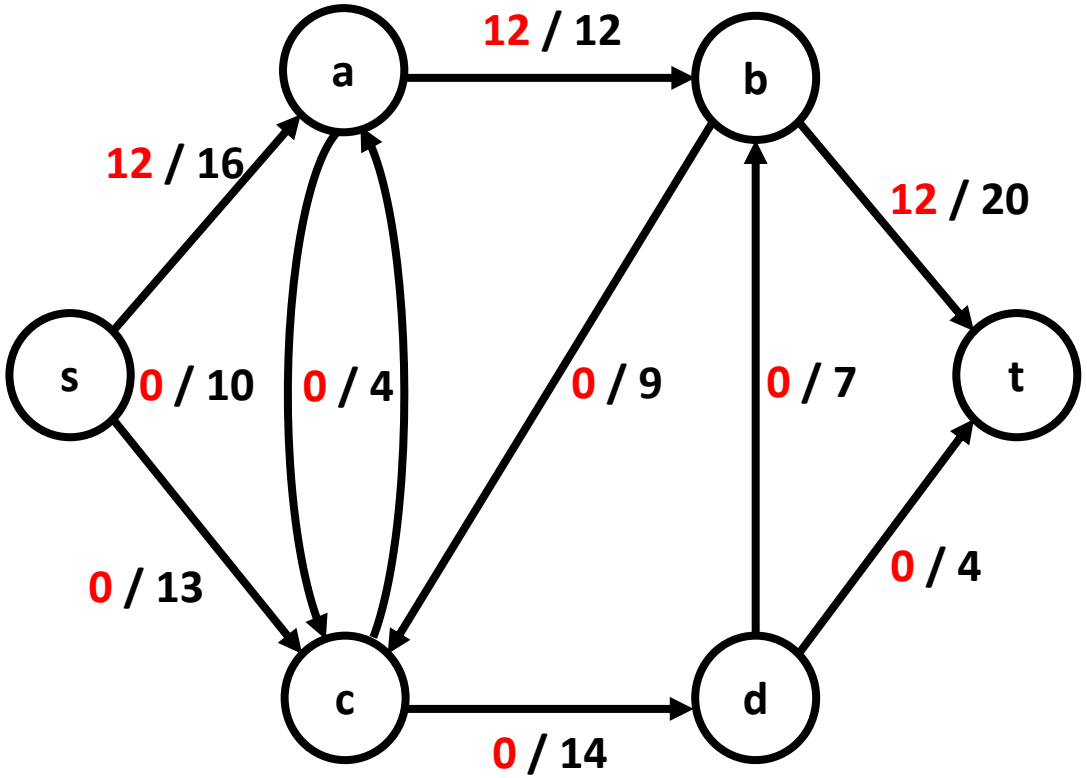
If $f(u, v) < c(u, v)$ for an edge $(u, v) \in E$, then $(u, v) \in E_f$ with $c_f(u, v) = c(u, v) – f(u, v) > 0$.

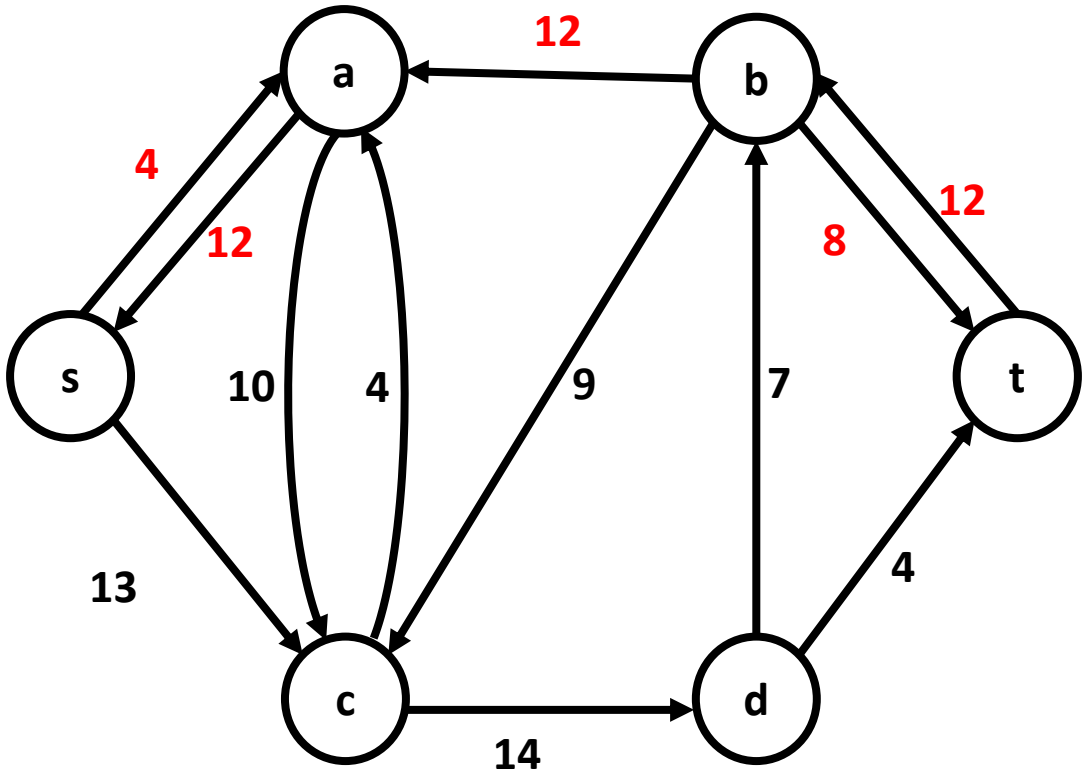
If $f(u, v) > 0$ for an edge $(u, v) \in E$, then $f(v, u) < 0$. Then, $(v, u) \in E_f$ with $c_f(v, u) = c(v, u) – f(v, u) > 0$.

If neither (u, v) nor (v, u) appears in E, then $c(u, v) = c(v, u) = 0$, $f(u, v) = f(v, u) = 0$. $c_f(u, v) = c_f(v, u) = 0$.

We conclude that an edge (u, v) can appear in a residual network only if at least one of (u, v) and (v, u) appears in the original network, and thus $|E_f| \leq 2|E|$.

Flow Network vs Residual Network





Augmenting Path

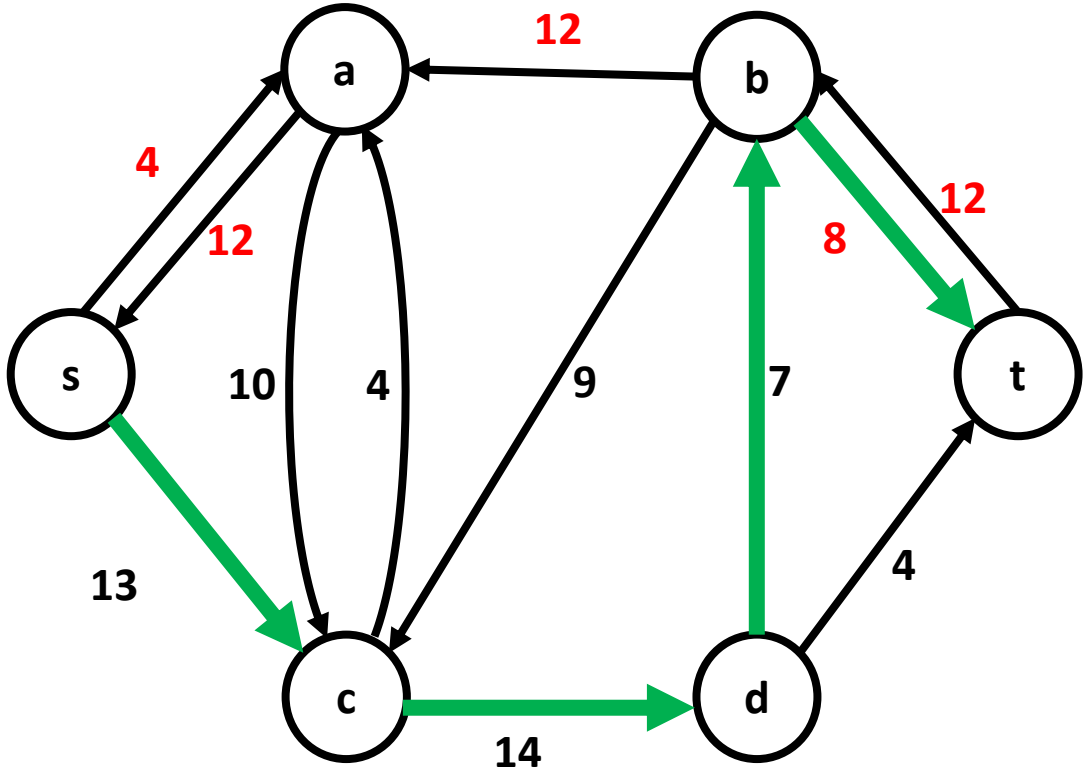
- Given a flow network G = (V, E) and a flow f , an **augmenting path** p is a simple path from s to t in the residual network G_f.

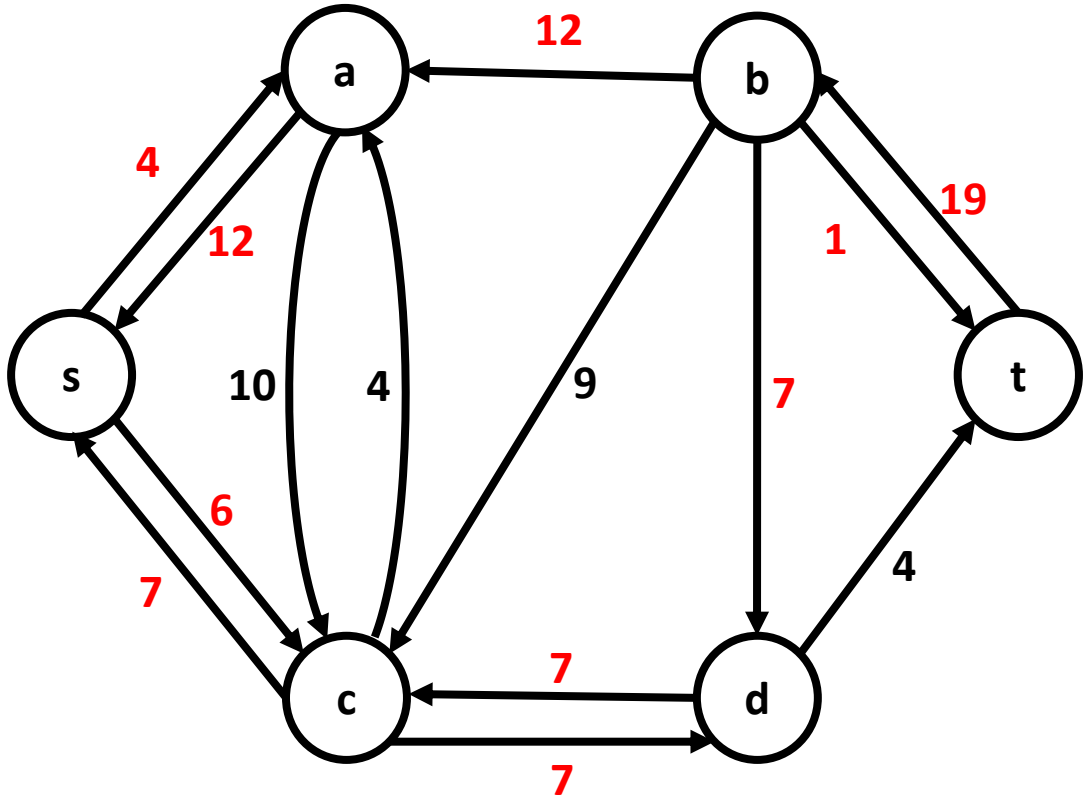
- By the definition of the residual network, each edge (u, v) on an augmenting path admits some additional positive flow from u to v without violating the capacity constraint on the edge.

- We call the maximum amount by which we can increase the flow on each edge in an augmenting path p, the **bottleneck capacity** of p, given by

$$c_f(p) = \min \{c_f(u, v) : (u, v) \text{ is on } p\}$$

Augmenting Path in Residual Network





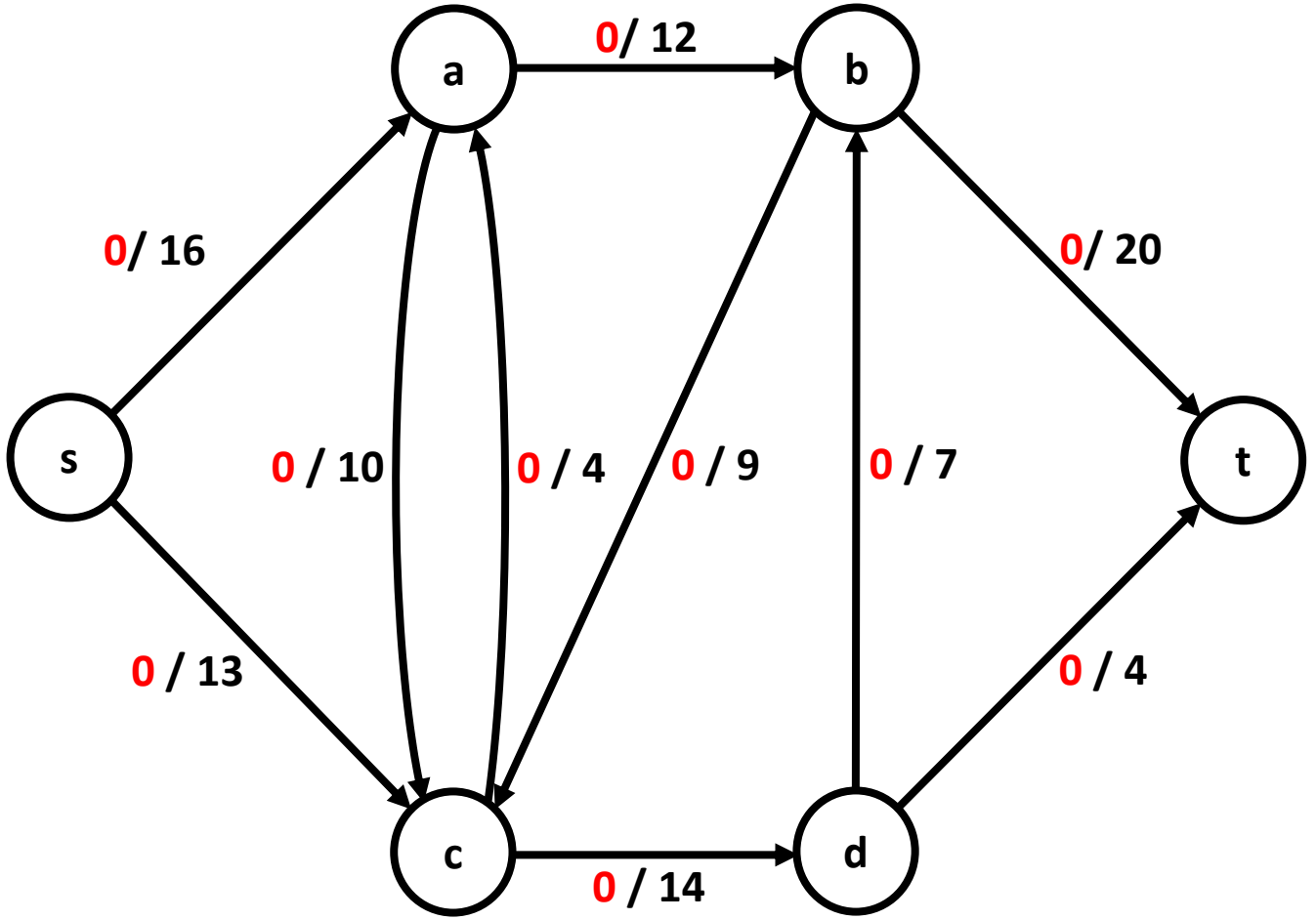
Bottleneck Capacity = min {13, 14, 7, 8} = 7

Ford-Fulkerson Algorithm

- Start with flow = 0.
- While there exists an augmenting path from s to t:
  - Find an augmenting path from s to t
  - Compute bottleneck capacity
  - Increase flow on that path by bottleneck capacity
  - Decrease capacity on that path by bottleneck capacity
- The variable flow gives the maximum flow.
- Run DFS to mark all reachable nodes from s. Call the set of vertices A.
- Cut edges of minimum size are from A to V – A.

Ford-Fulkerson Algorithm

Initialize Flow: Start with 0 flow.

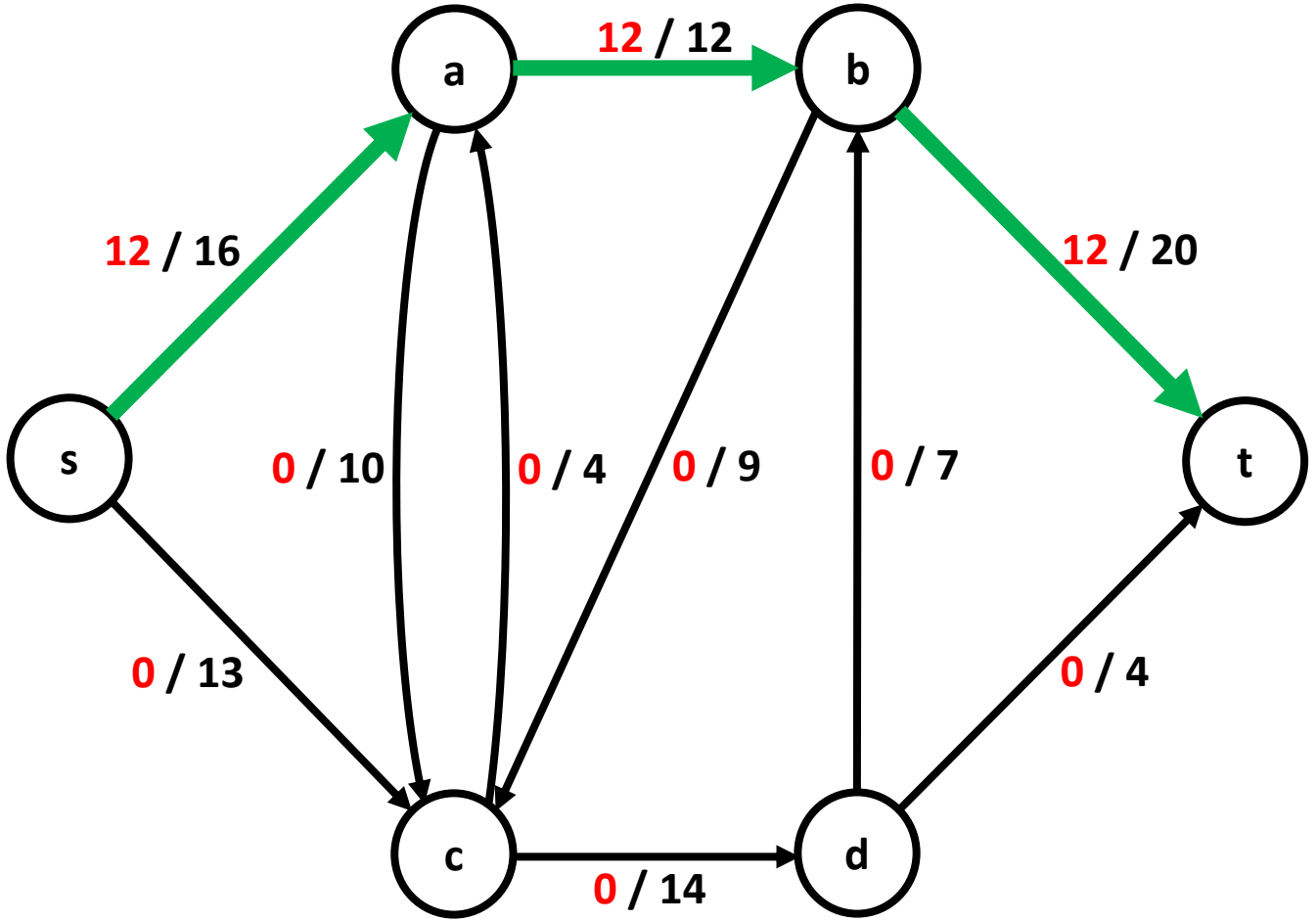


Flow = 0

Ford-Fulkerson Algorithm

Augmenting path. Find an undirected path from *s* to *t* such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



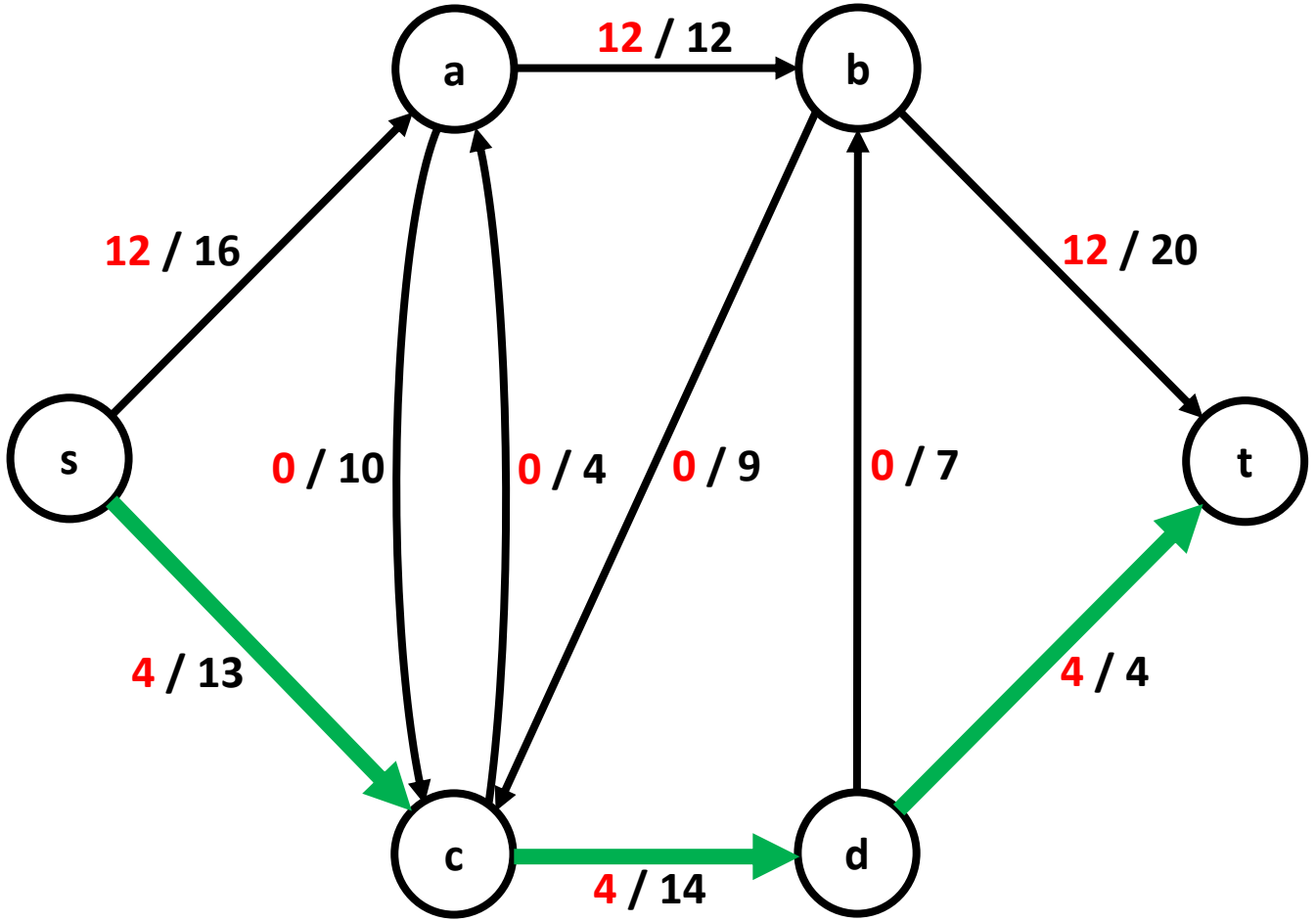
Bottleneck Capacity = 12

Flow = 0 + 12 = 12

Ford-Fulkerson Algorithm

Augmenting path. Find an undirected path from *s* to *t* such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



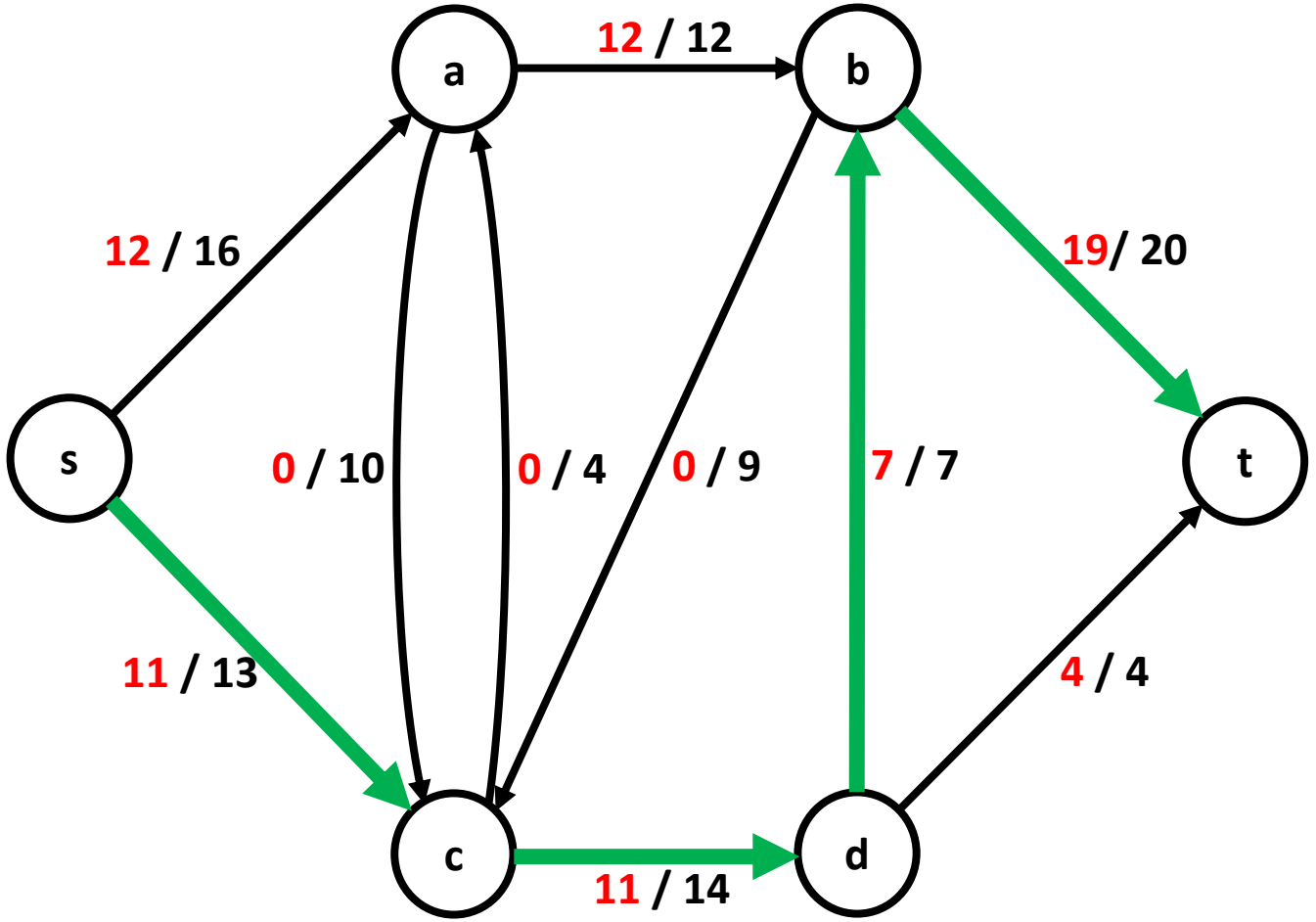
Bottleneck Capacity = 4

Flow = 0 + 12 + 4 = 16

Ford-Fulkerson Algorithm

Augmenting path. Find an undirected path from *s* to *t* such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).



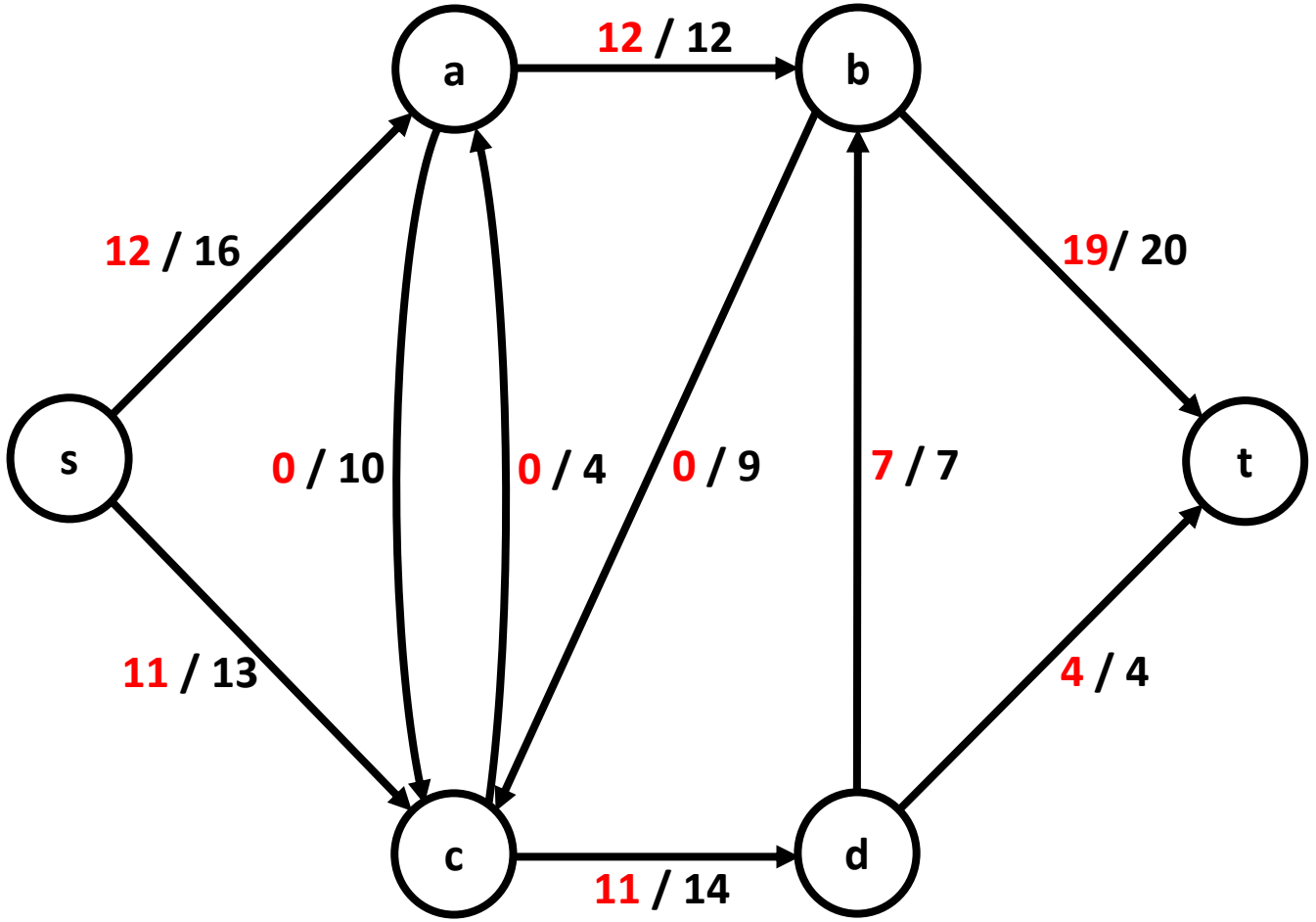
Bottleneck Capacity = 7

Flow = 0 + 12 + 4 + 7 = 23

Ford-Fulkerson Algorithm

Termination (No Augmenting path). All paths from *s* to t are blocked by either a

- Full forward edge.
- Empty backward edge.

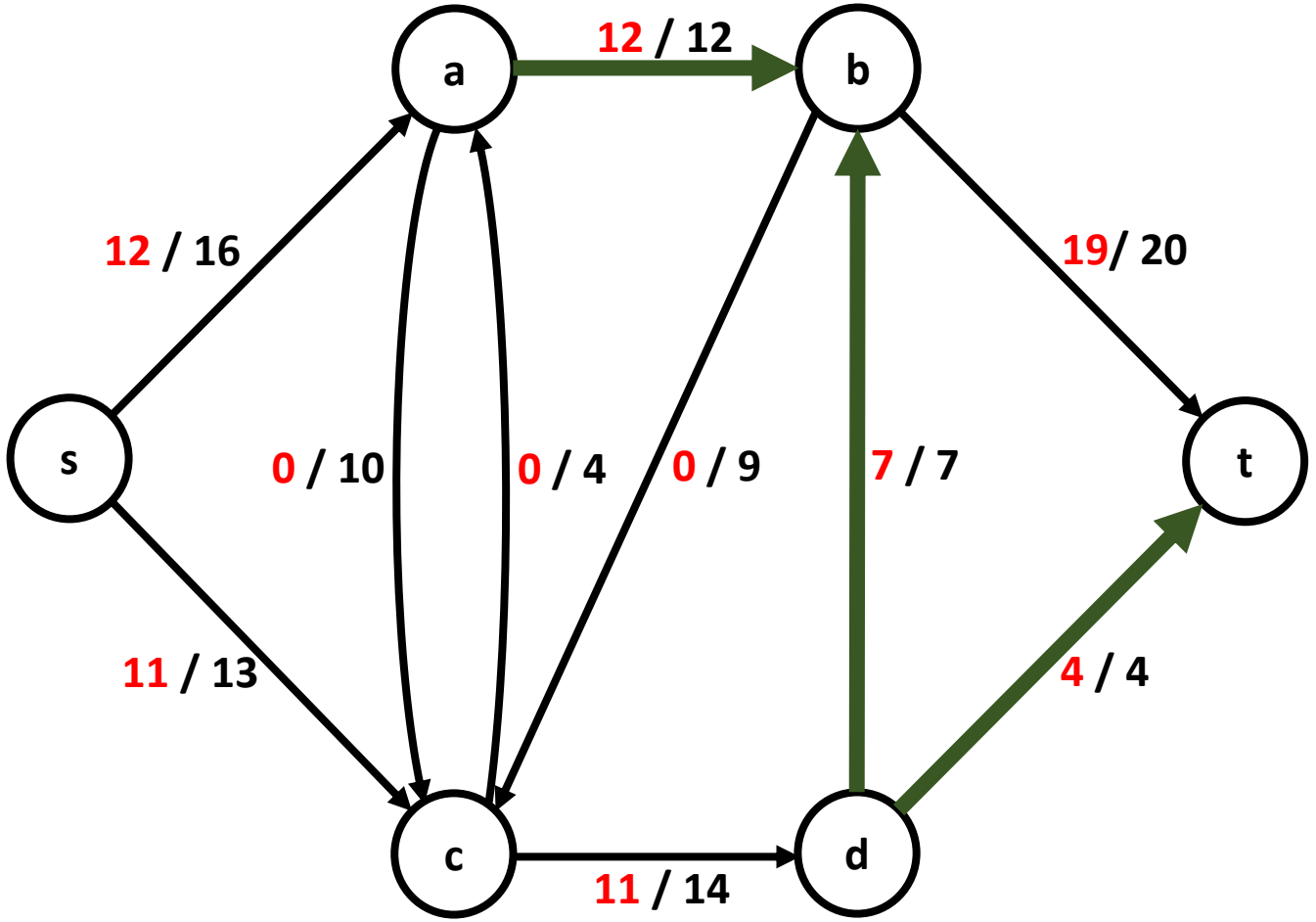


Bottleneck Capacity = 7

Flow = 0 + 12 + 4 + 7 = 23

Ford-Fulkerson Algorithm

Minimum Cut: DFS on residual graph



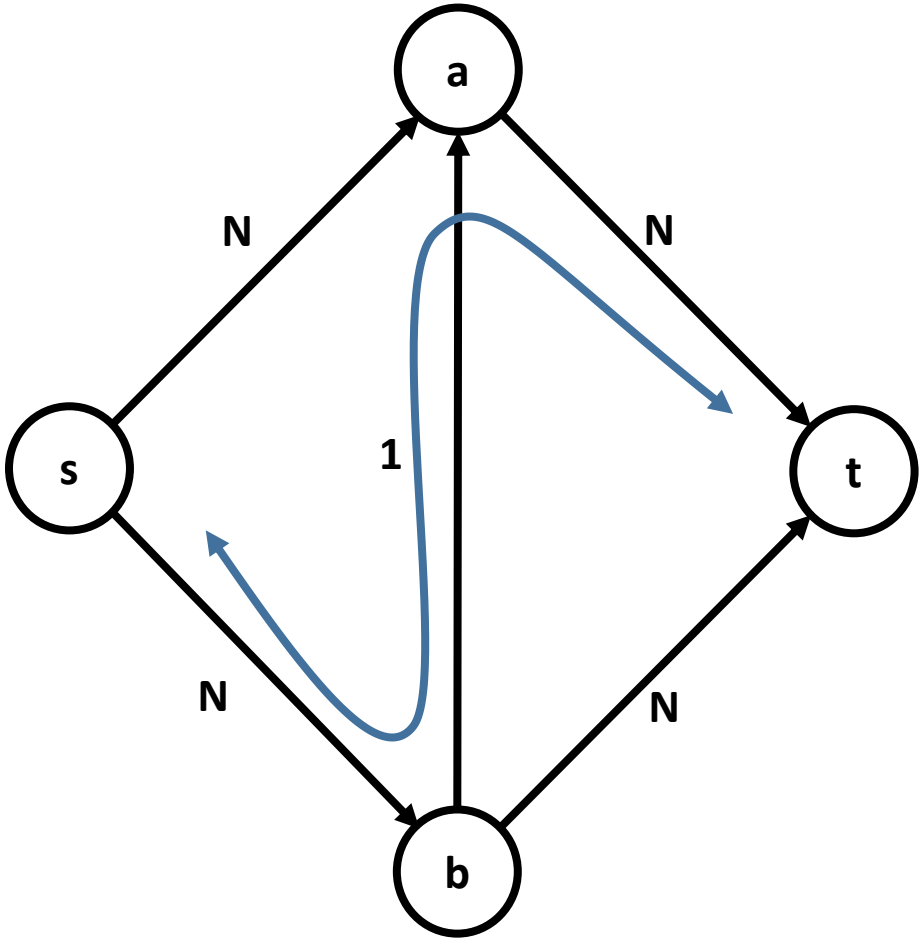
Cut = 12 + 7 + 4 = 23

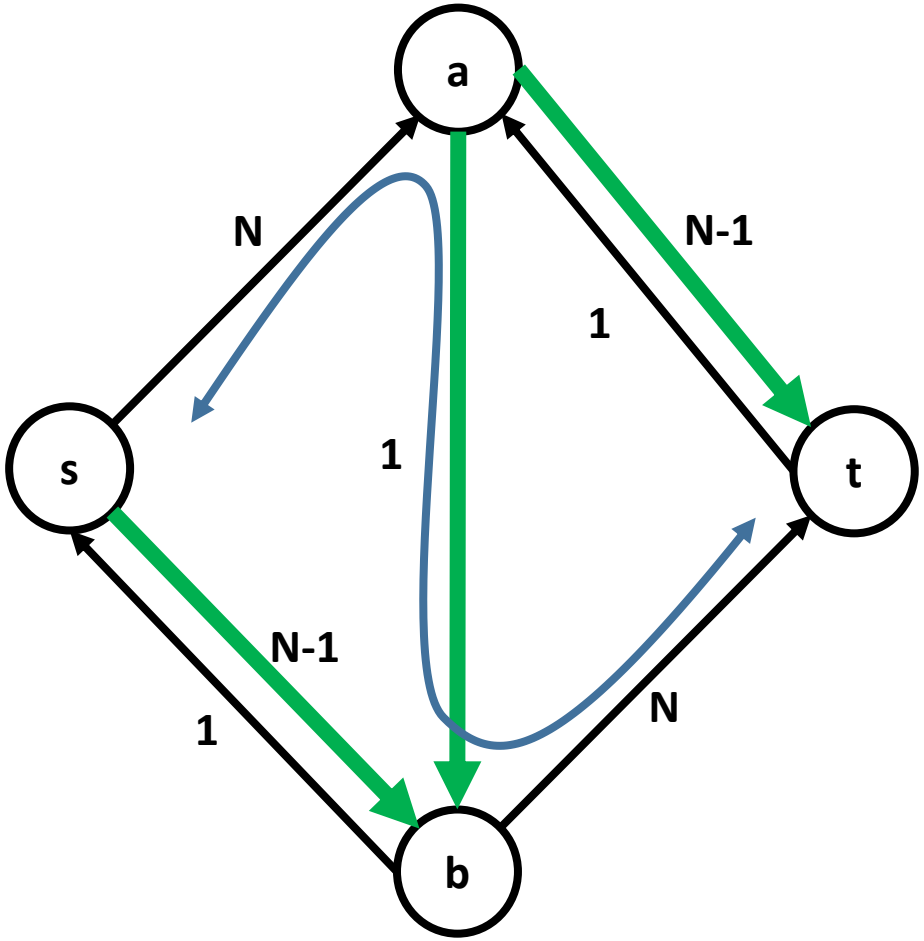
Ford-Fulkerson Algorithm: Running Time

- Start with flow = 0. O(|V| + |E|)
- While there exists an augmenting path from s to t: O(f*)
  - Find an augmenting path using DFS from s to t O(|V| + |E|)
  - Compute bottleneck capacity
  - Increase flow on that path by bottleneck capacity
  - Decrease capacity on that path by bottleneck capacity
- The variable flow gives the maximum flow.
- Run DFS to mark all reachable nodes from s. Call the set of vertices A. O(|V| + |E|)
- Cut edges of minimum size are from A to V – A.

Ford-Fulkerson Algorithm: A Bad Case

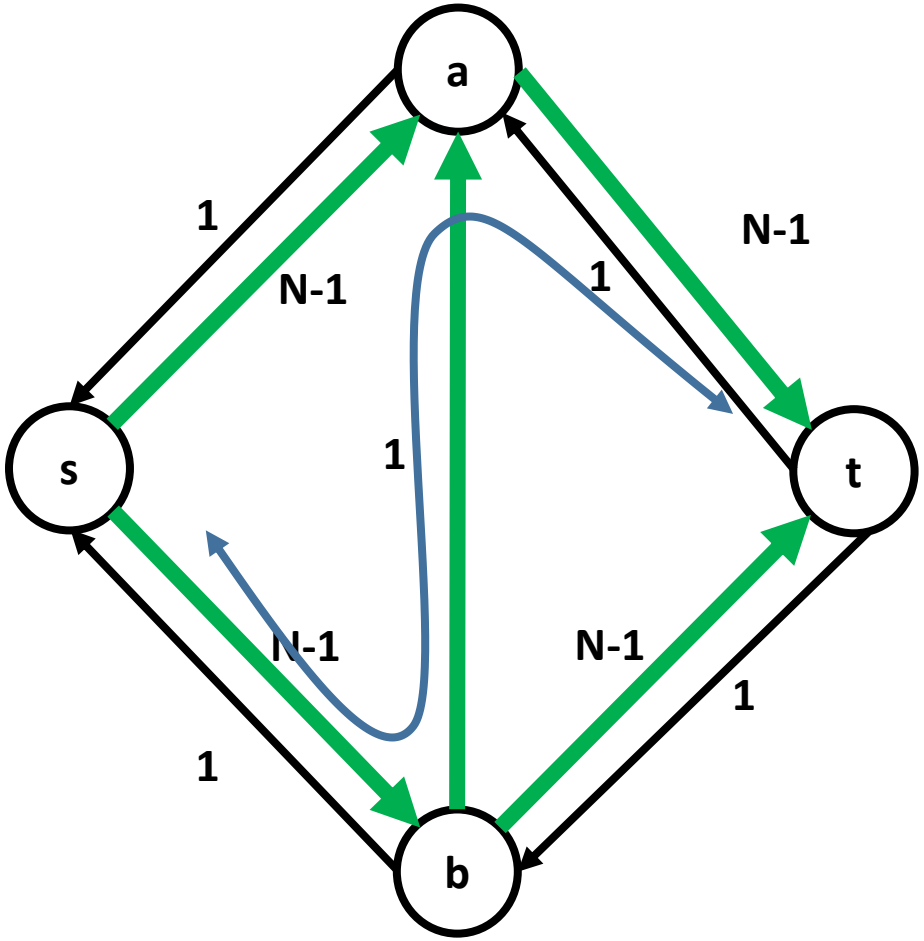
Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maximum flow.

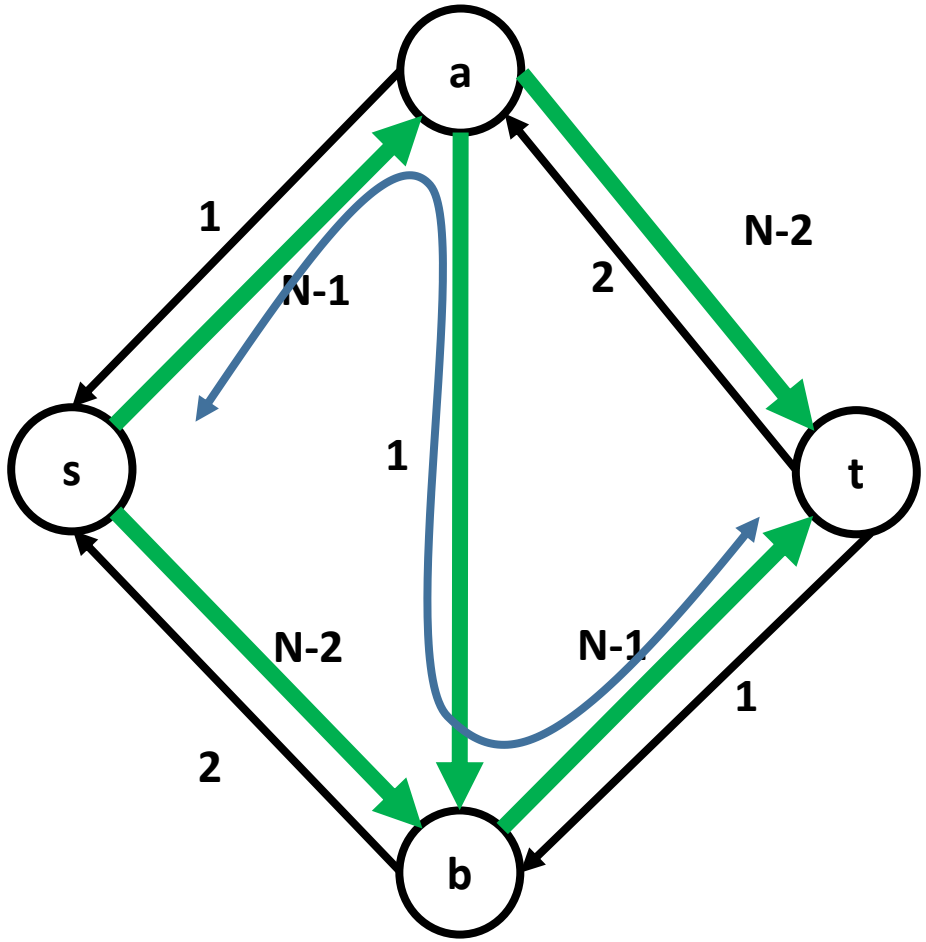




Ford-Fulkerson Algorithm: A Bad Case

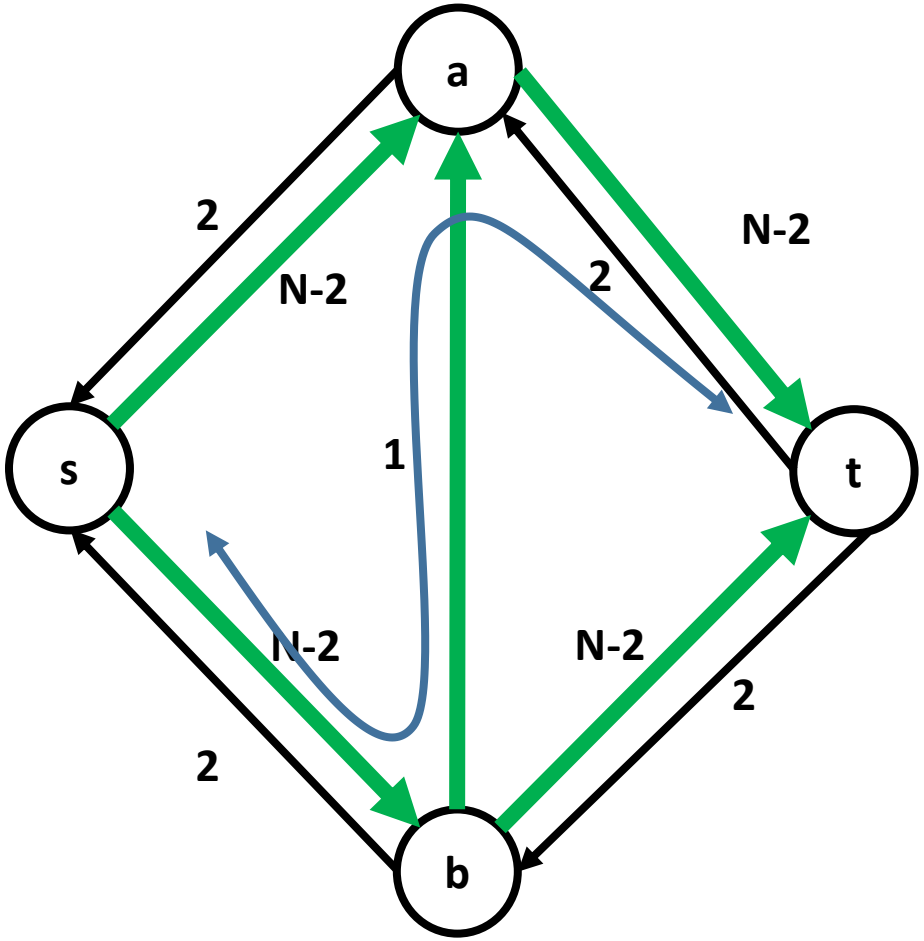
Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maximum flow.

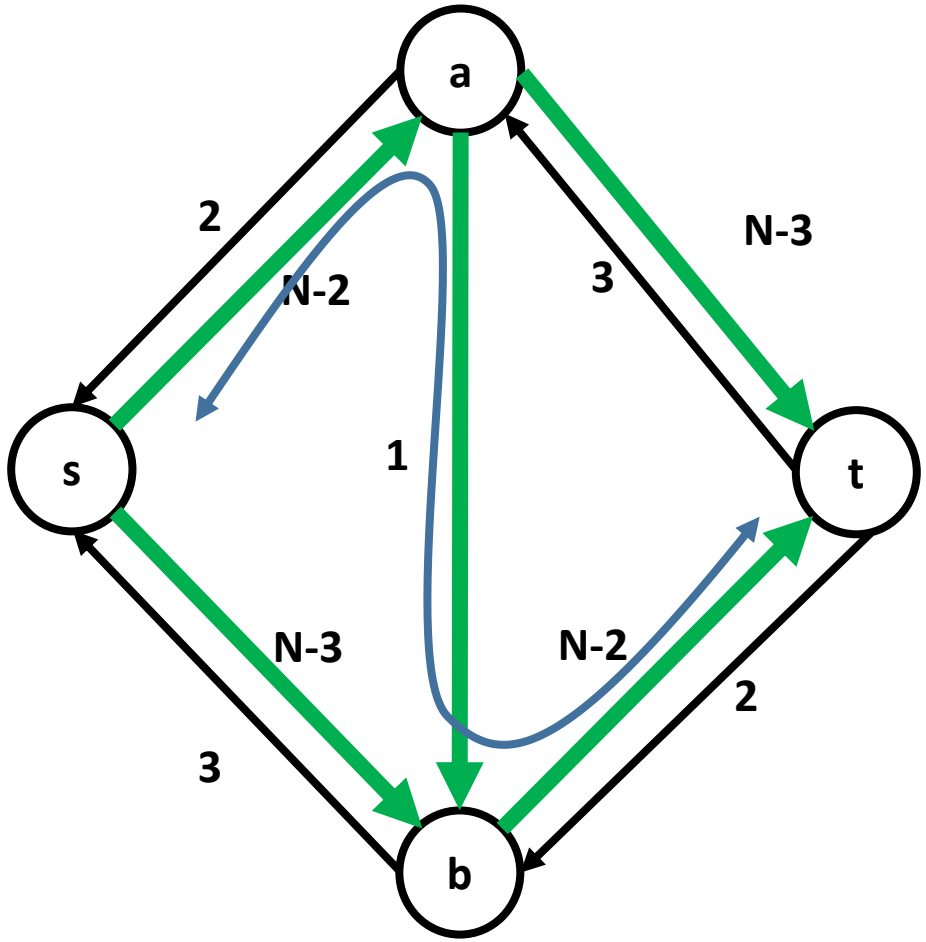




Ford-Fulkerson Algorithm: A Bad Case

Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maximum flow.



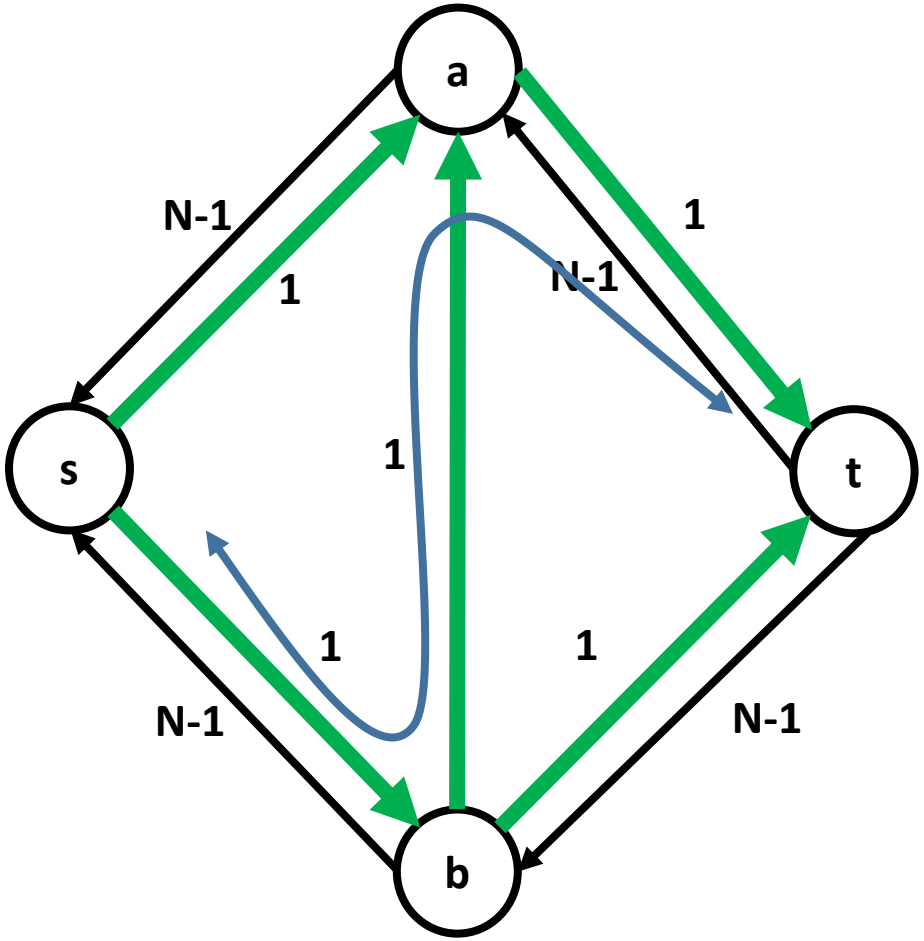


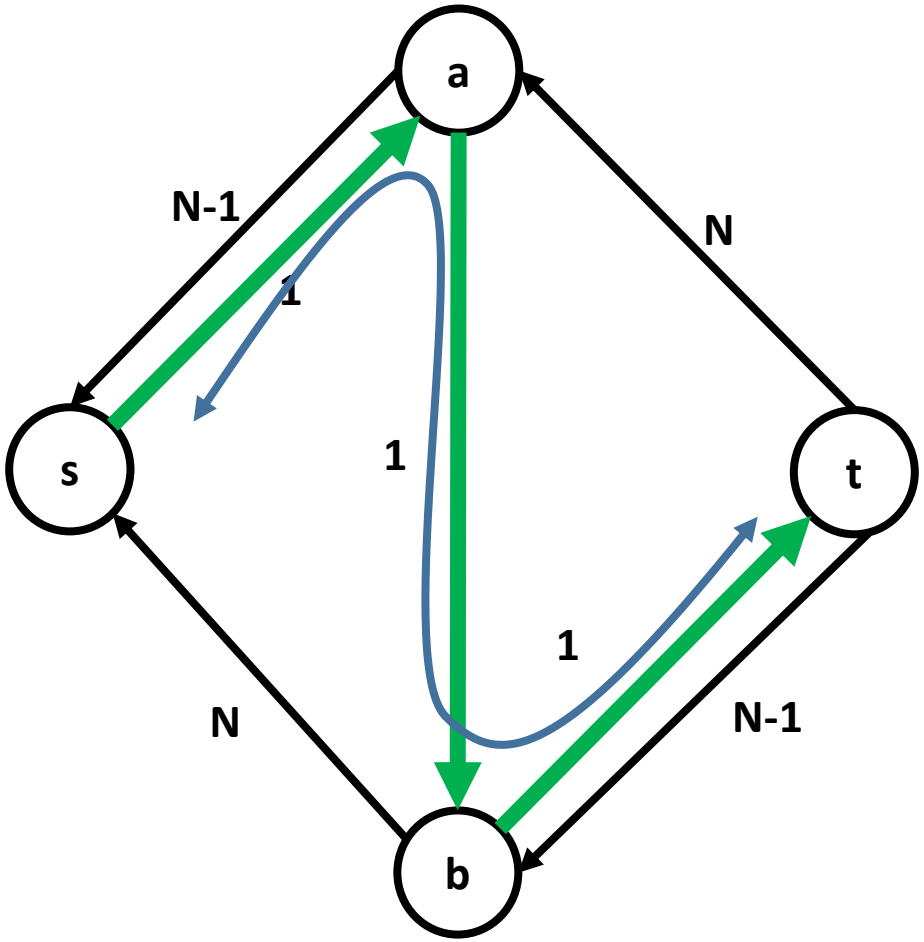
Ford-Fulkerson Algorithm: A Bad Case

Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maximum flow.

Ford-Fulkerson Algorithm: A Bad Case

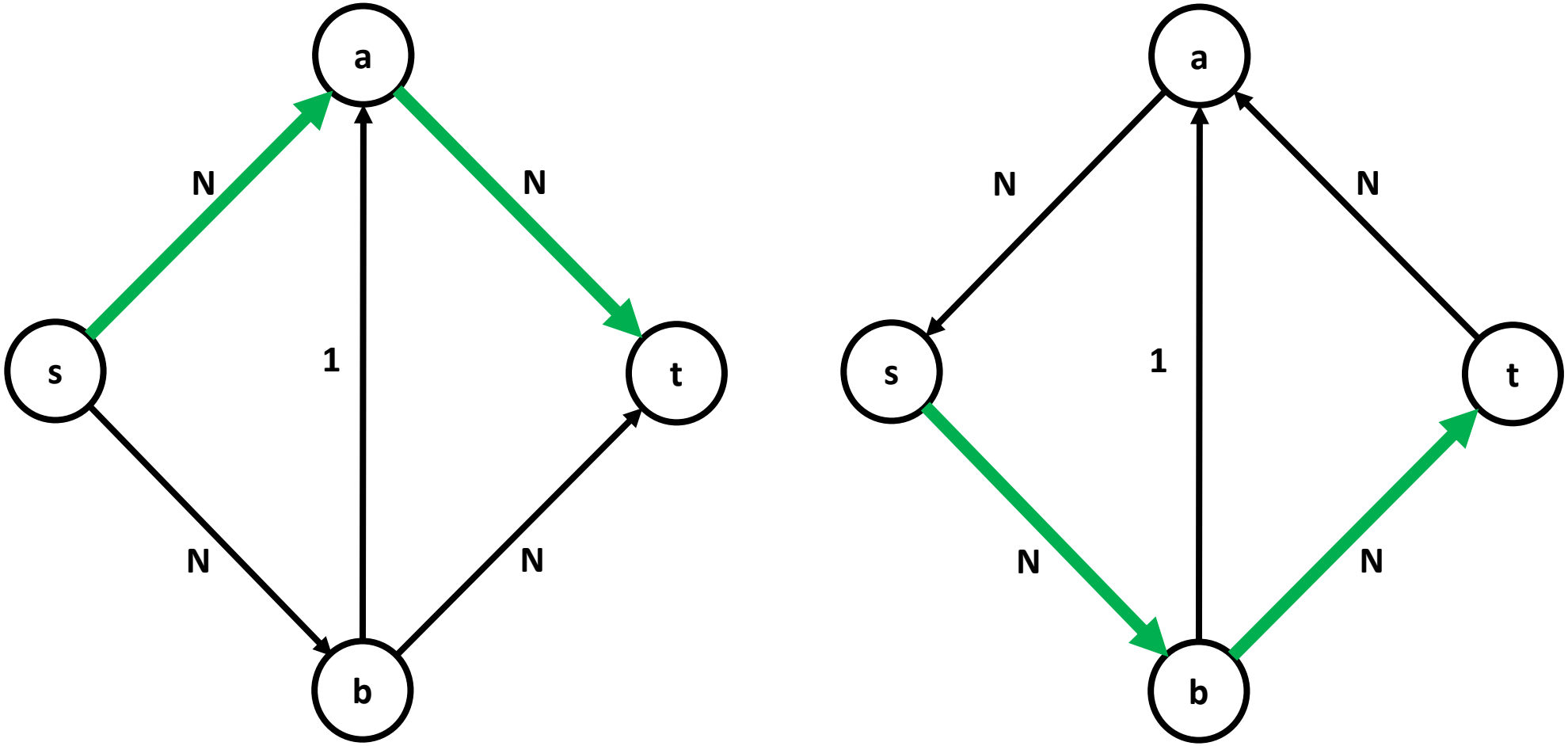
Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maximum flow.





Ford-Fulkerson Algorithm: Good News

This case can easily be avoided by using shortest path. Finding augmenting path by BFS from s to t.



Ford-Fulkerson Method

- It is a method proposed by Ford and Fulkerson, where they have suggested to find an augmenting path from s to t.
- It is not an algorithm since they do not mention about the exact way to find the augmenting path.
- DFS: O(f*(|V| + |E|))
- BFS: O(|V||E|(|V| + |E|)) [Edmond]

- Implementation Note:

  Represent the residual network as an adjacency list.