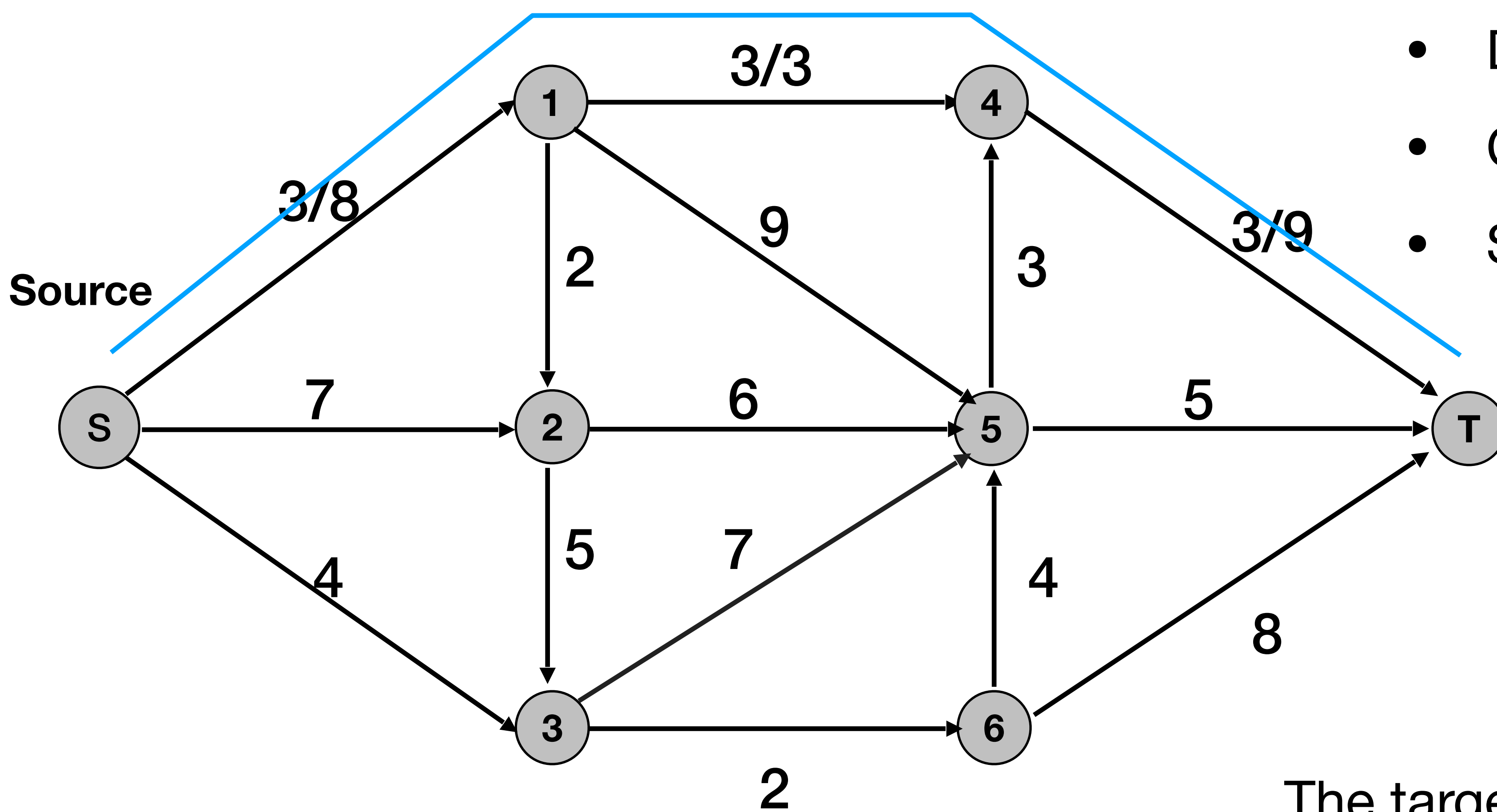
Max Flow and Min Cut

Maximum Flow

Network: abstraction from rail road.

- Directed graph
- Capacities on edges
- Source S and sink T



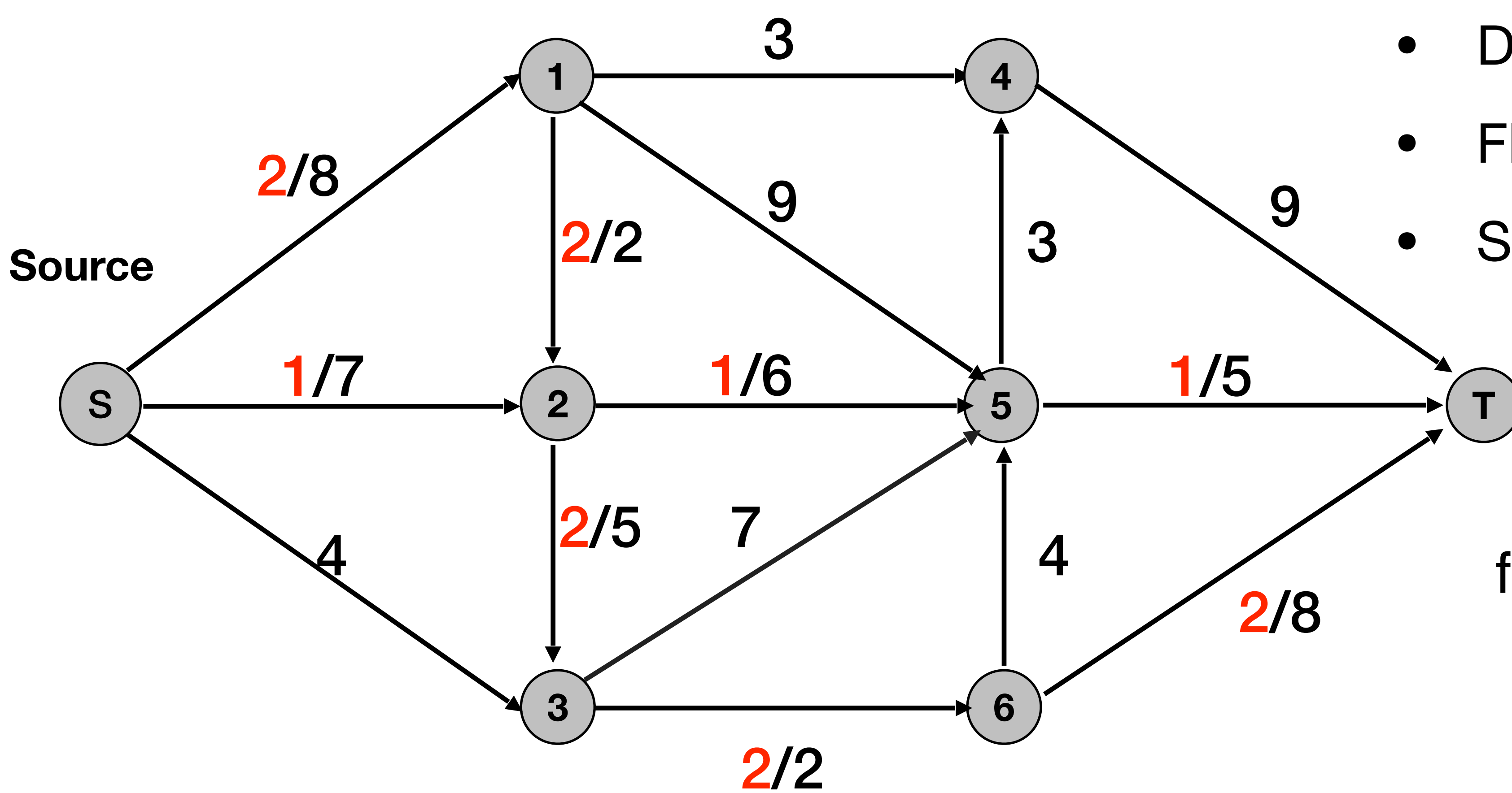
"Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other"

T.E. Harris

The target is to transport freight from source to sink.

A flow is an assignment of weights of cargo/freight to edges (roads).

Maximum Flow



Network: abstraction from rail road.

- Directed graph
- Flow capacities on edges
- Source S and sink T

flow f(s,t) = 2+1 =3

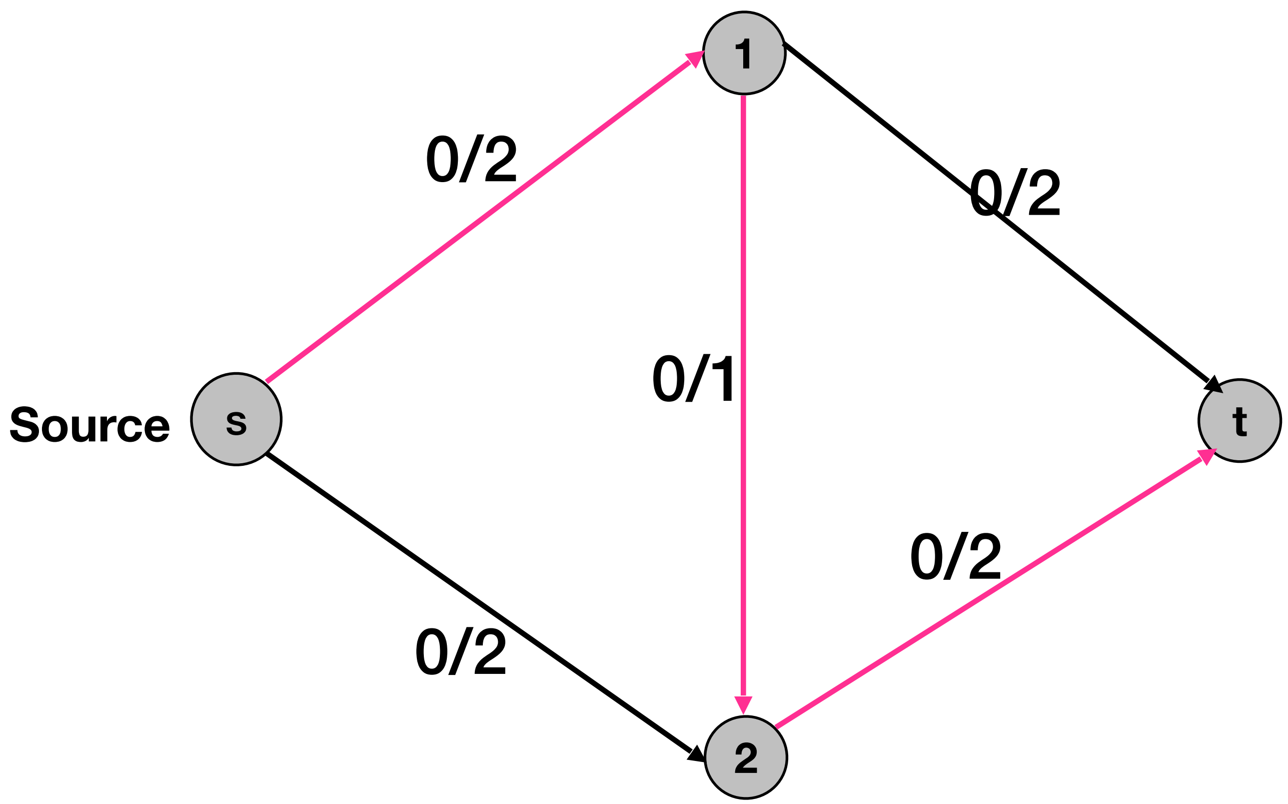
Max flow problem: Assign **flow** to edges so as to:

1. Flow on an edge cannot exceed edge capacity.   E.g., flow on (2,3) = 1 < Capacity 2
2. For any intermediate vertex, inflow = outflow.   E.g., Vertex 2, inflow = 3

outflow = 1+2=3

Target: maximize flow sent from source s to sink t

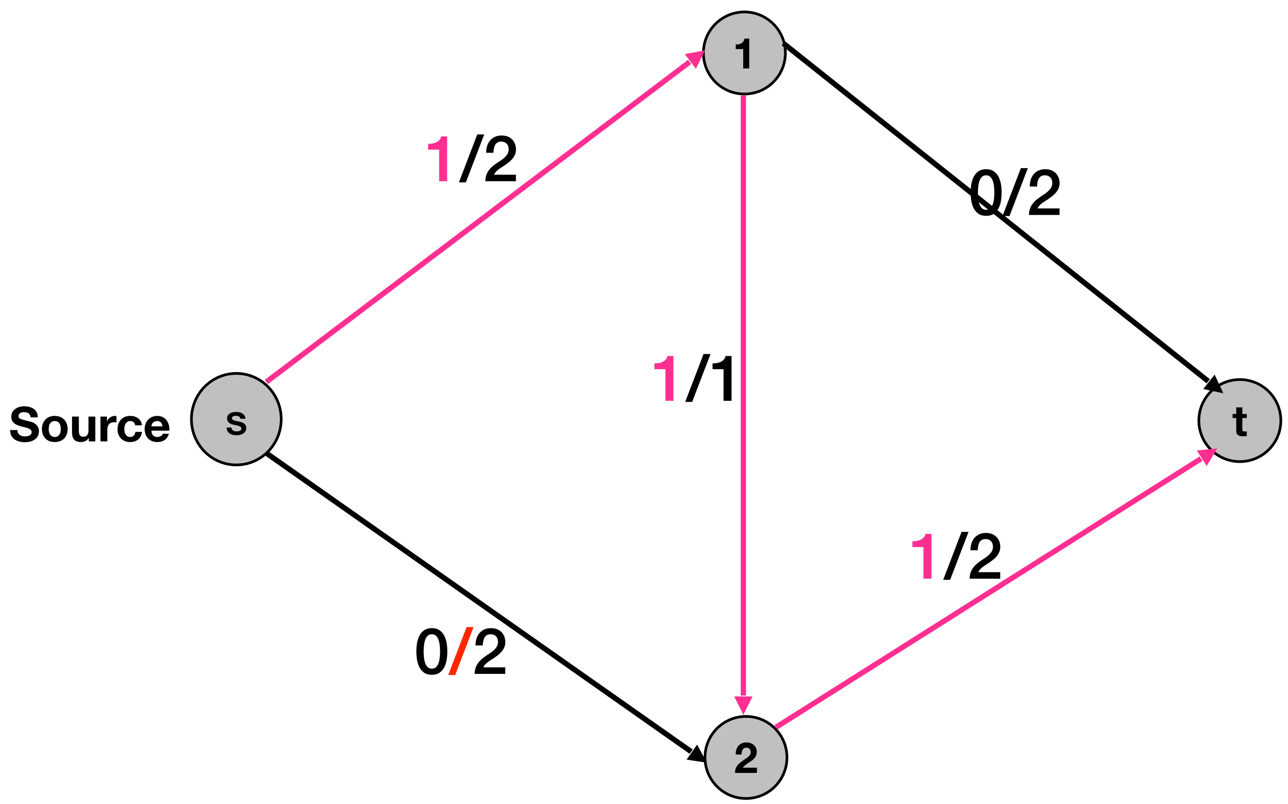
Greedy algorithm may fail on G



Greedy algorithm:

- Find an s→t path where each edge has flow(e) < capacity(e).
- Augment flow along path P.
- Repeat until you get stuck.

Greedy algorithm may fail on G



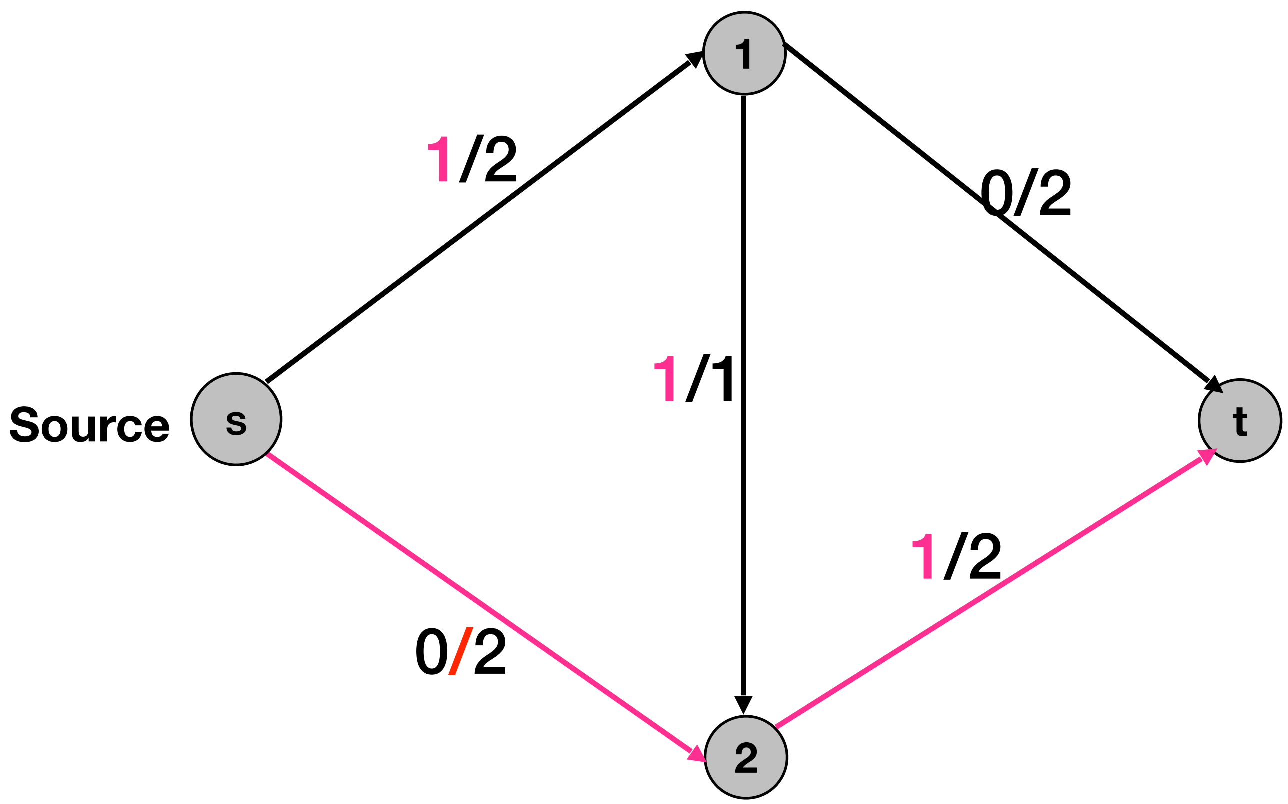
The minimum capacity of edges on P is 1.

The current path P could send at most 1

Greedy algorithm:

- Find an s→t path where each edge has flow(e) < capacity(e).
- Augment flow along path P.
  - Compute the maximum flow x path P could send
  - Increase flow of edges on path P by x
- Repeat until you get stuck.

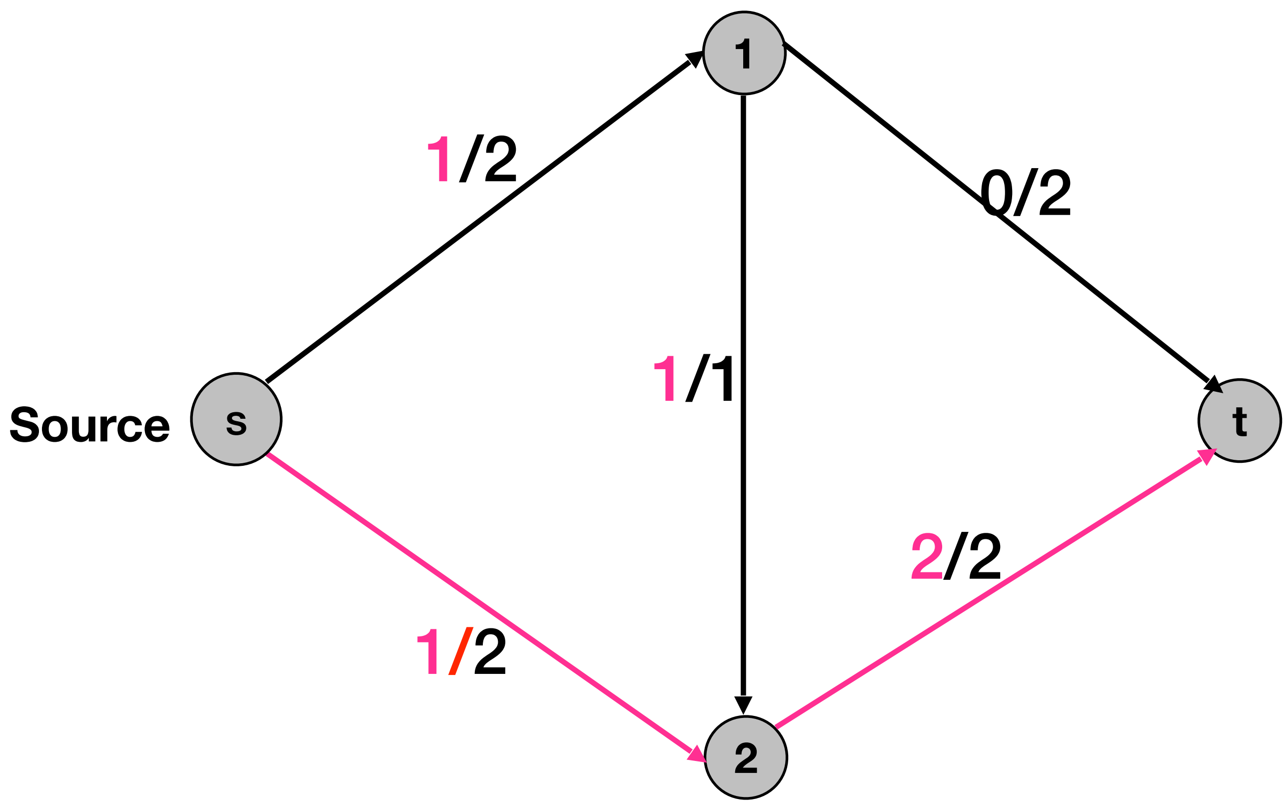
Greedy algorithm may fail on G



Greedy algorithm:

- Find an s→t path where each edge has flow(e) < capacity(e).
- Augment flow along path P.
  - Compute the maximum flow x path P could send
  - Increase flow of edges on path P by x
- Repeat until you get stuck.

Greedy algorithm may fail on G



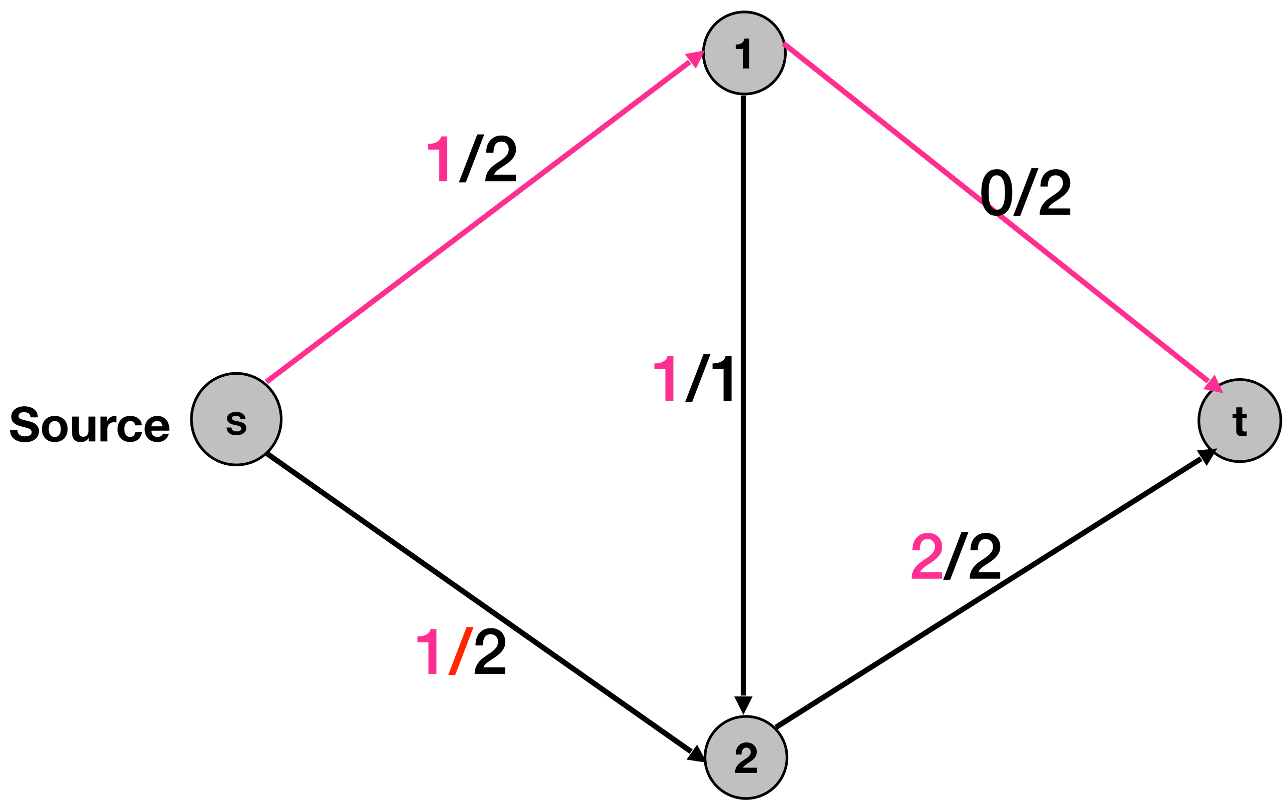
The minimum capacity of edges on P is 1.

The current path P could send at most 1

Greedy algorithm:

- Find an s→t path where each edge has flow(e) < capacity(e).
- Augment flow along path P.
  - Compute the maximum flow x path P could send
  - Increase flow of edges on path P by x
- Repeat until you get stuck.

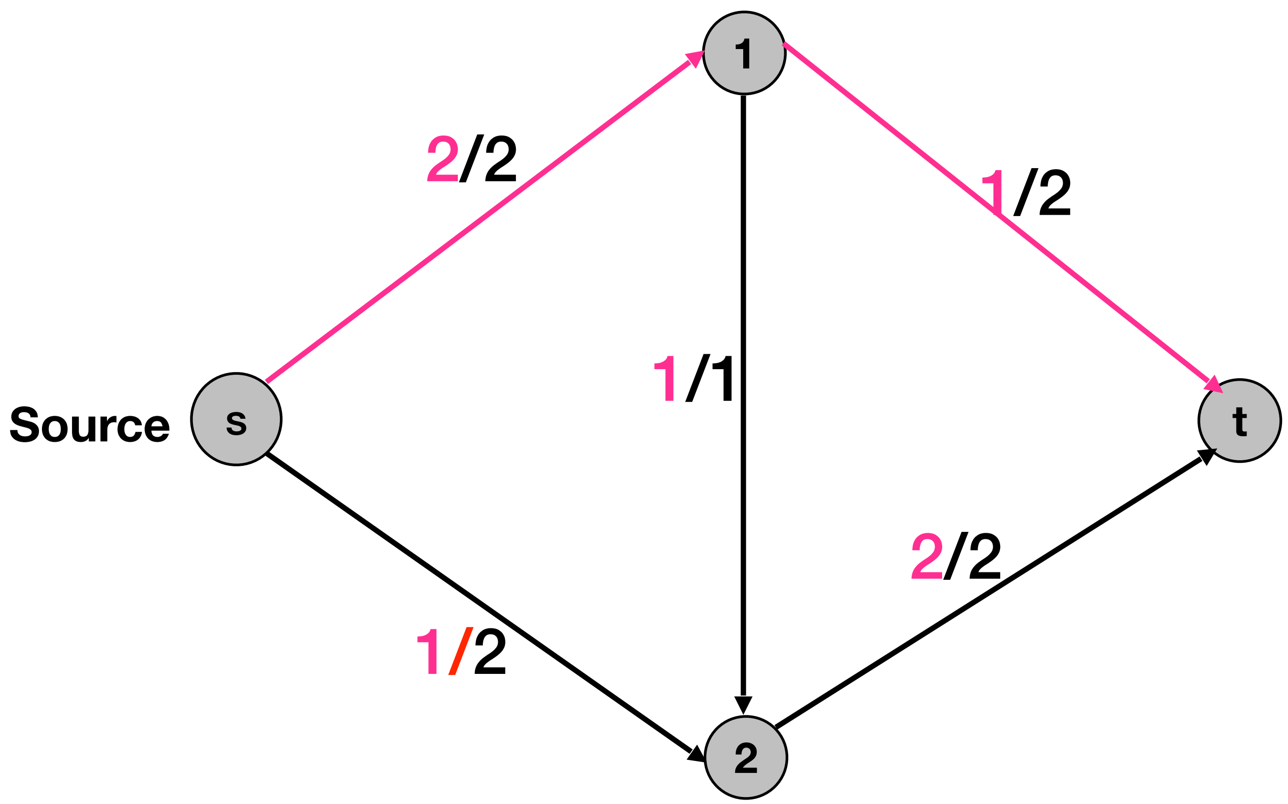
Greedy algorithm may fail on G



Greedy algorithm:

- Find an s→t path where each edge has flow(e) < capacity(e).
- Augment flow along path P.
  - Compute the maximum flow x path P could send
  - Increase flow of edges on path P by x
- Repeat until you get stuck.

Greedy algorithm may fail on G



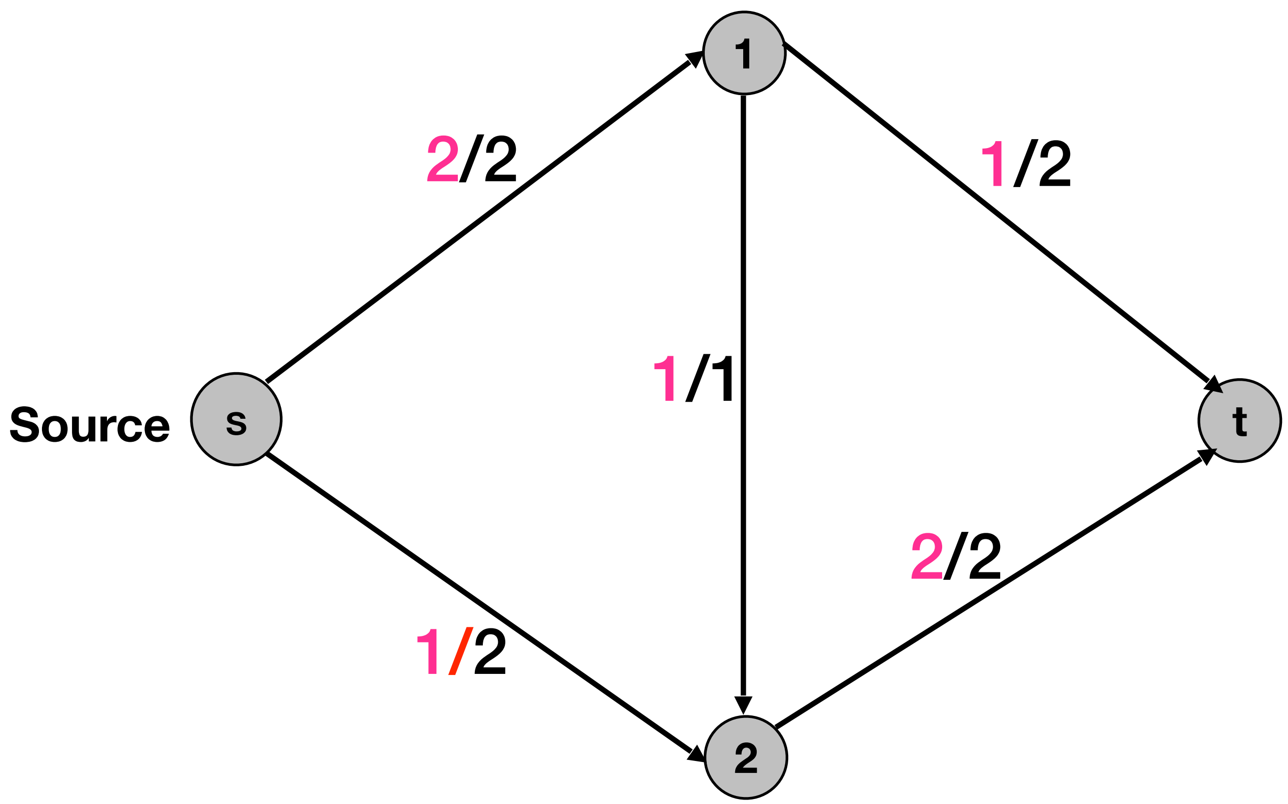
The minimum capacity of edges on P is 1.

The current path P could send at most 1

Greedy algorithm:

- Find an s→t path where each edge has flow(e) < capacity(e).
- Augment flow along path P.
  - Compute the maximum flow x path P could send
  - Increase flow of edges on path P by x
- Repeat until you get stuck.

Greedy algorithm may fail on G



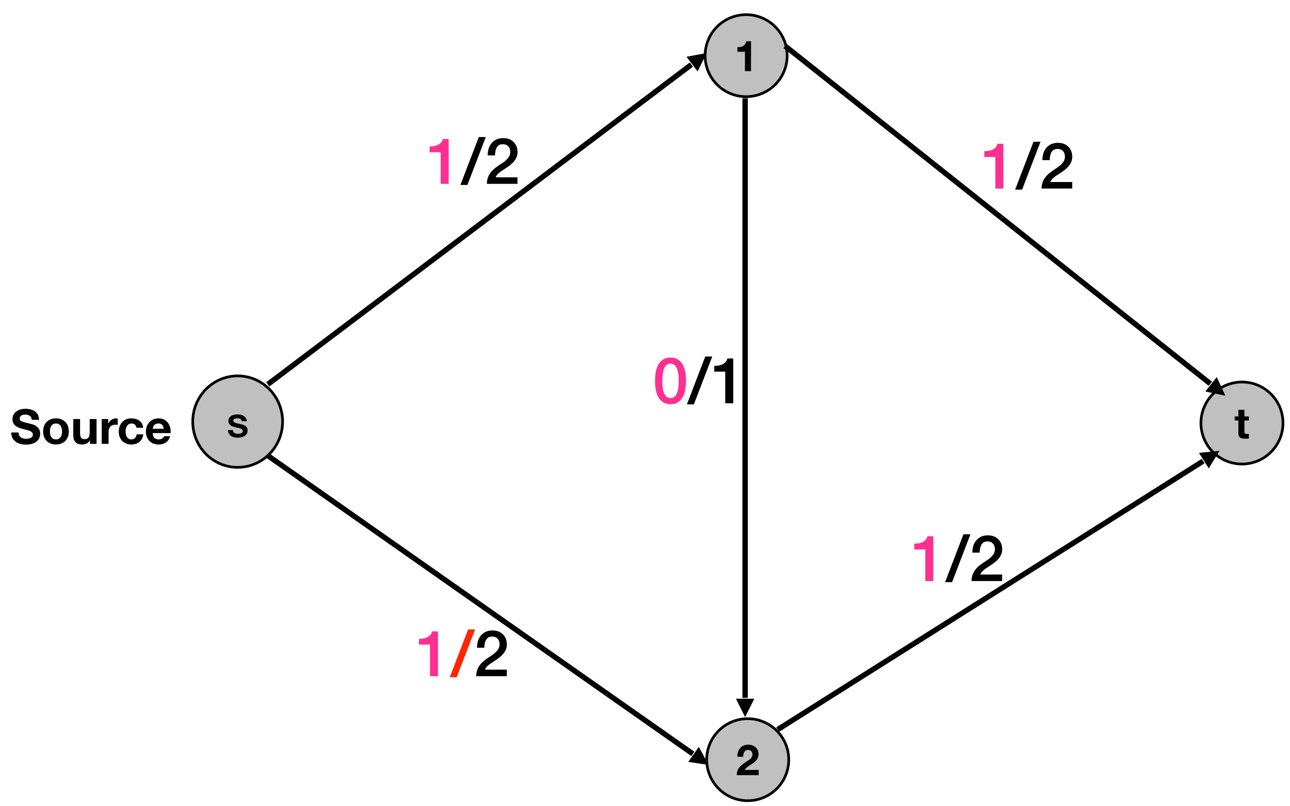
The maximum flow computed is 3.

But the answer is 4.

Greedy algorithm:

- Find an s→t path where each edge has flow(e) < capacity(e).
- Augment flow along path P.
  - Compute the maximum flow x path P could send
  - Increase flow of edges on path P by x
- Repeat until you get stuck.

Augment the flow to achieve the maximum



The maximum flow computed is 3.

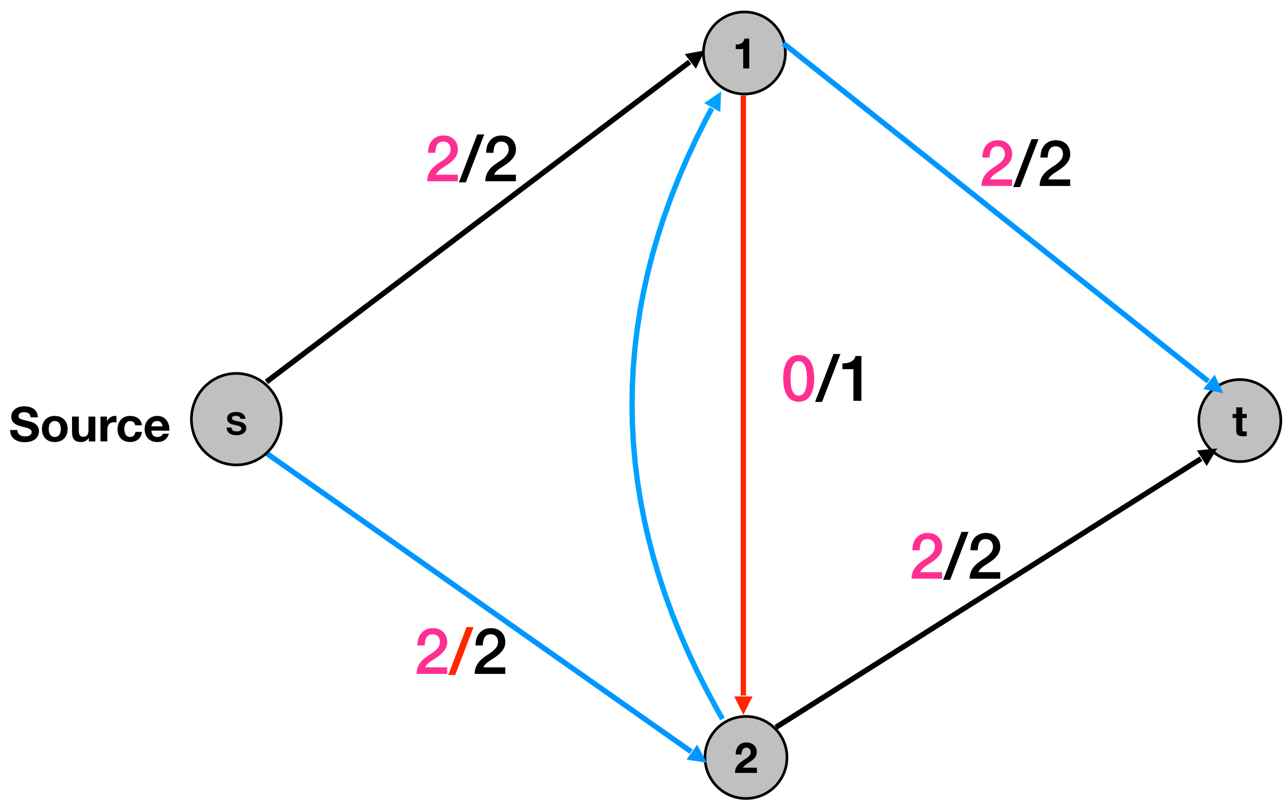
But the answer is 4.

Decrement the flow on Path(s, 1, 2, t)

Resend the flow through Path(s, 1, t)

Increment the flow on Path(s, 2, t)

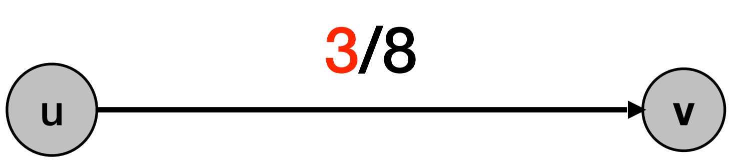
An easy way to augment the flow



We need to add a reverse flow from 2 to 1
and allow flow 1 sent through s, 2, 1 and t.

Residual Graph

Edge of Original Graph:
flow(u,v) = 3
capacity(u,v) = 8

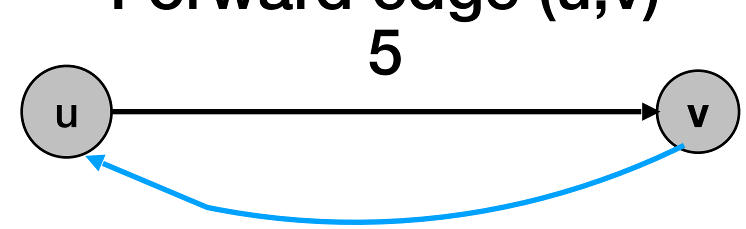


Forward edge

Transform

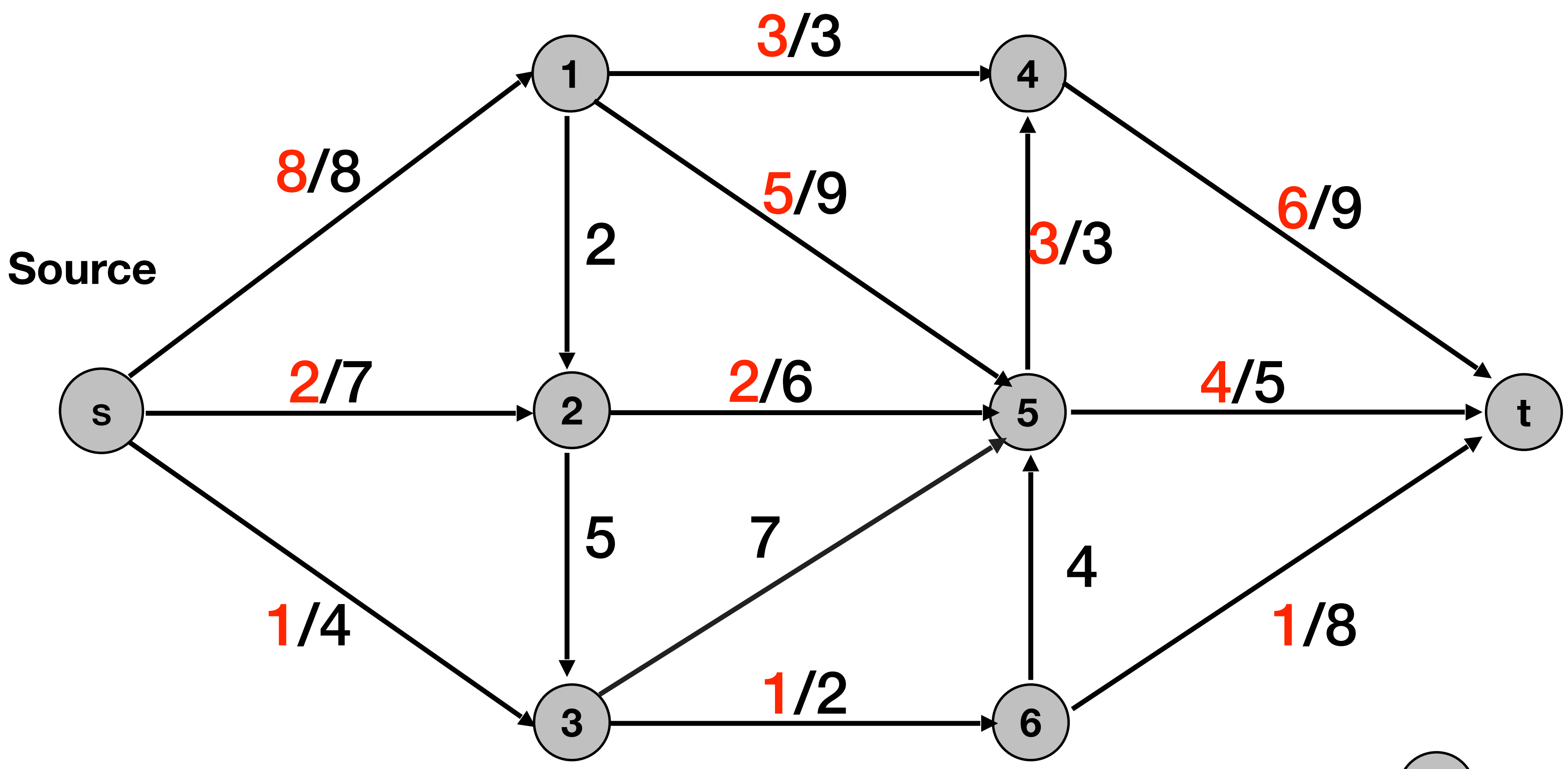
Edge of Residual Graph:
Residual Capacity of (u,v) = capacity(u,v)-flow(u,v)
Capacity of backward edge (v) = flow(u,v)

Forward edge (u,v)



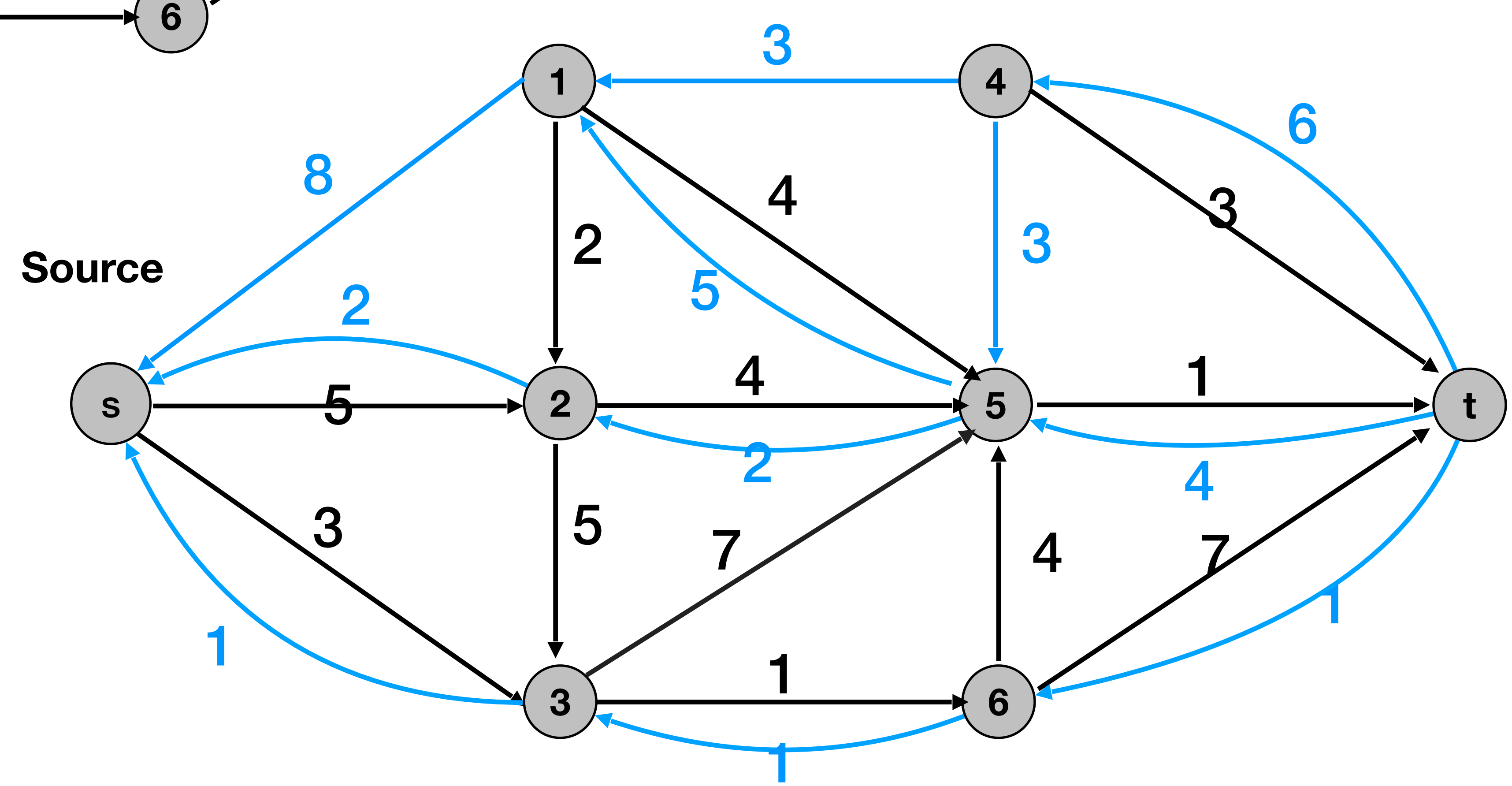
Backward edge (v, u)

Residual Graph



Original Graph

Residual Graph



Ford-Fulkerson Algorithm

FORD–FULKERSON(G)

For each edge $e \in E$: $flow(e) \leftarrow 0$.

$G_r \leftarrow$ residual network of G with respect to flow f.

WHILE (there exists an s↝t path P in G_r) // breath first search

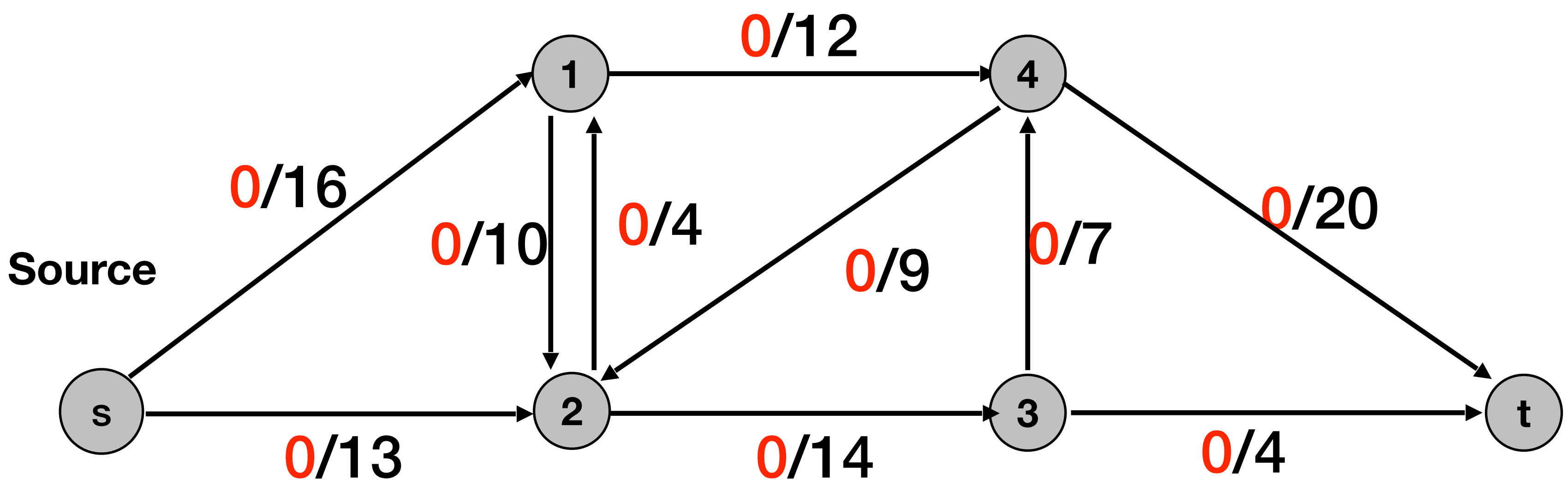
$f = f$ +AUGMENT(f, c, P). //augmenting flow on path P

Update G_r.

RETURN f.

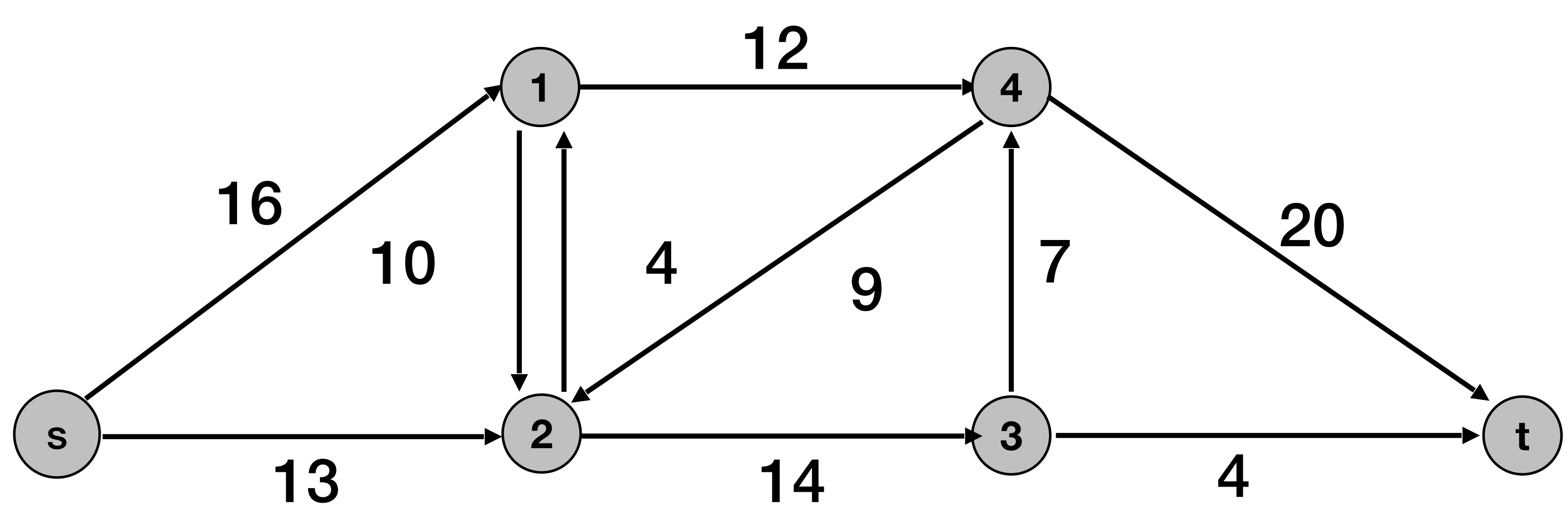
AUGMENT(f, c, P**)**
{
Compute the minimum residual flow c **of edges of P**
Increase the flow of edges on P by c.
}

Ford-Fulkerson Algorithm

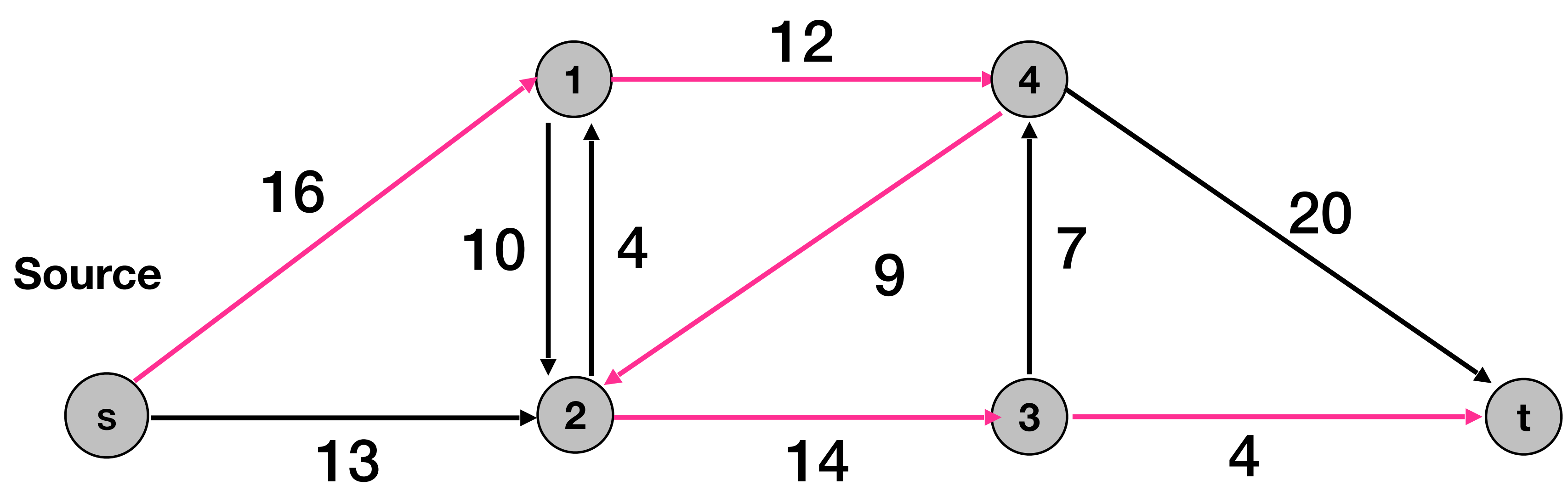


Original Graph

Residual Graph



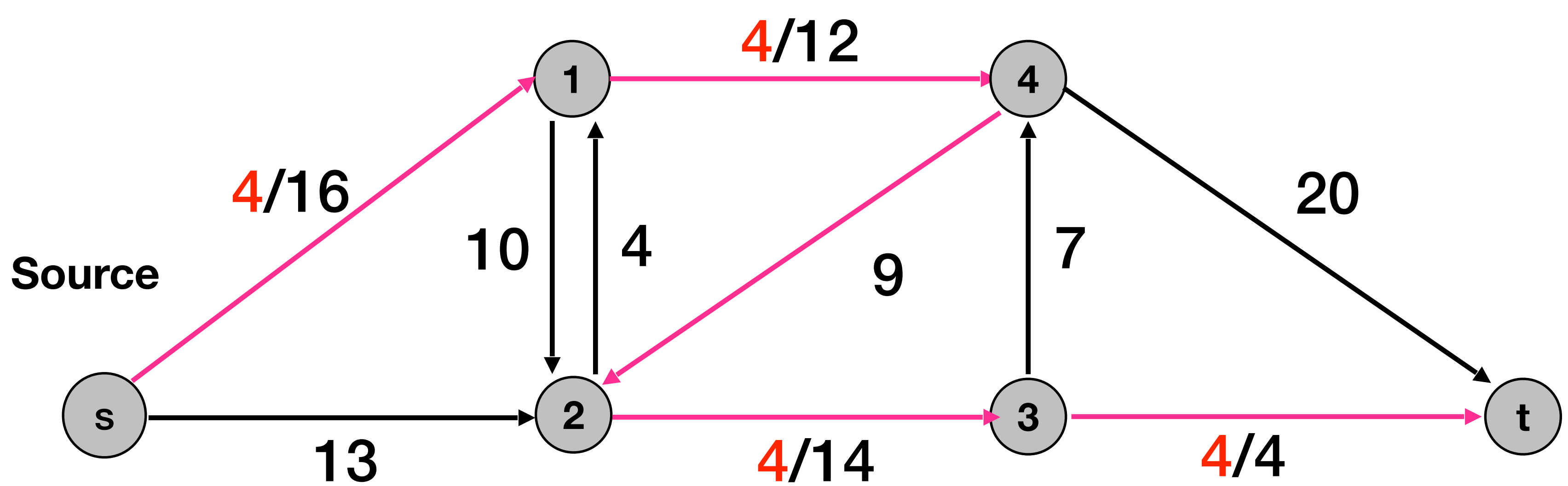
Ford-Fulkerson Algorithm



Residual Graph Gr

`Use BFS to find a path P from s to t on Gr`

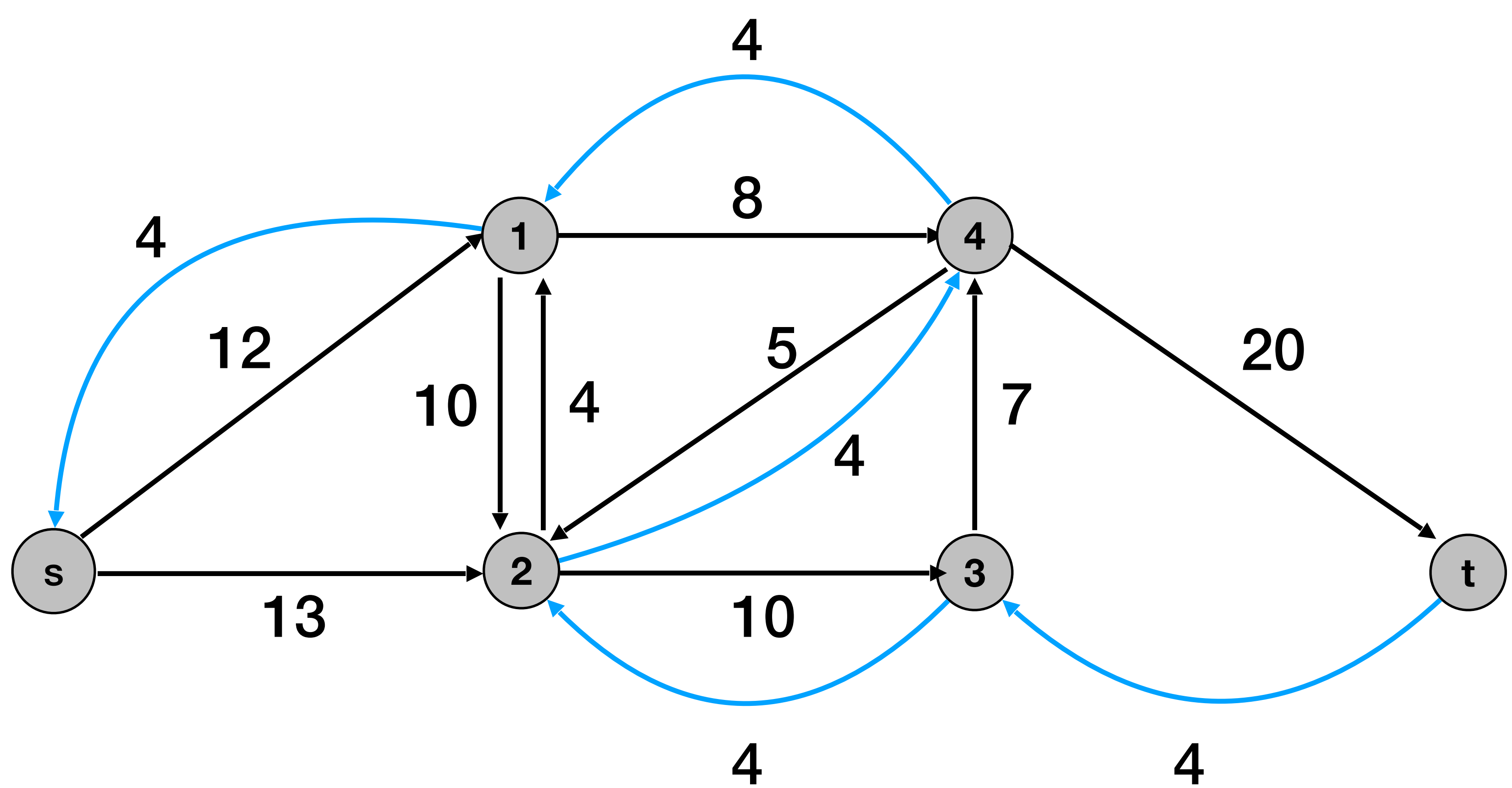
Ford-Fulkerson Algorithm



Residual Graph Gr

Augment the flow on P of Gr

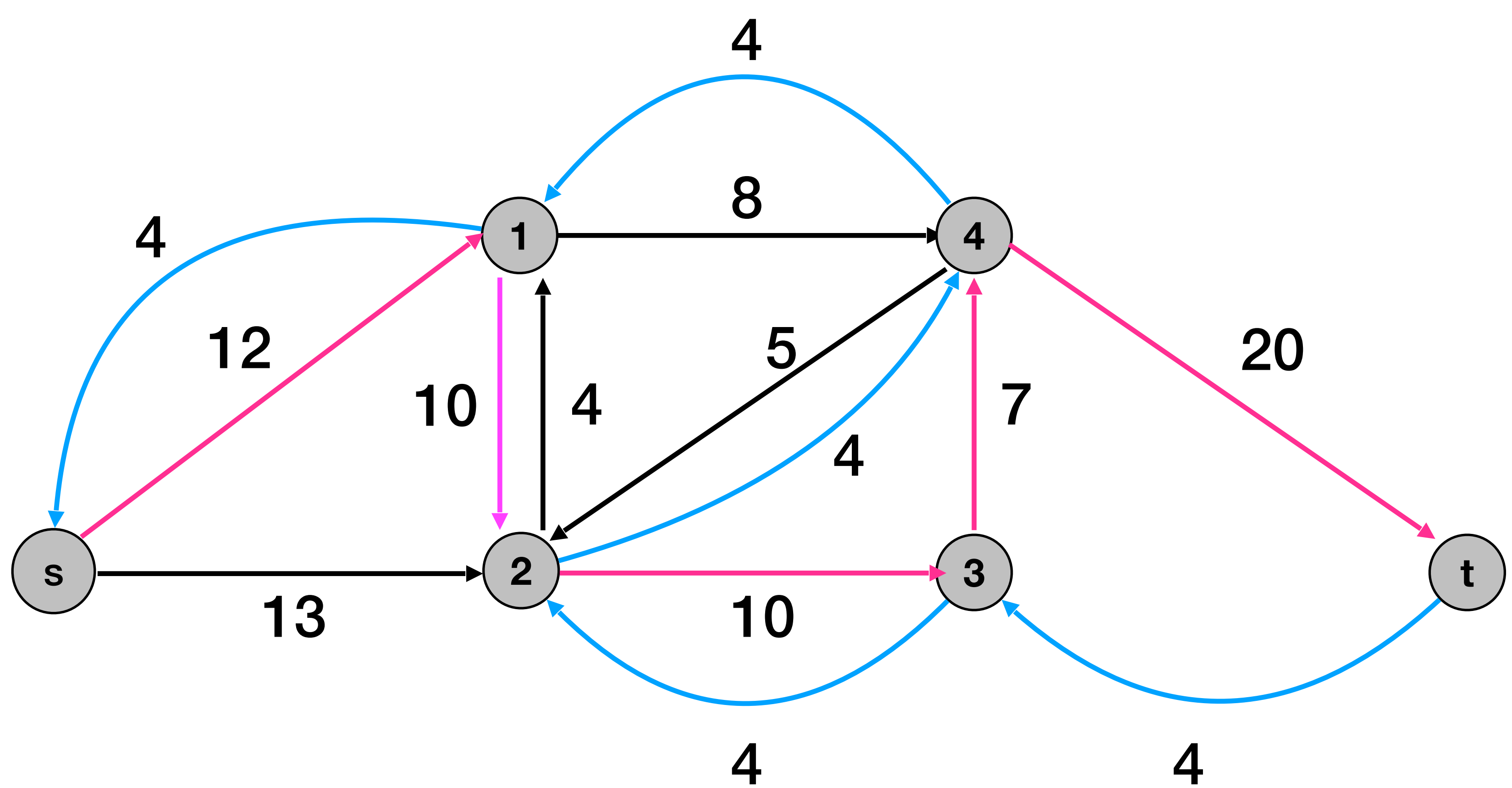
Ford-Fulkerson Algorithm



Residual Graph Gr

Update Gr

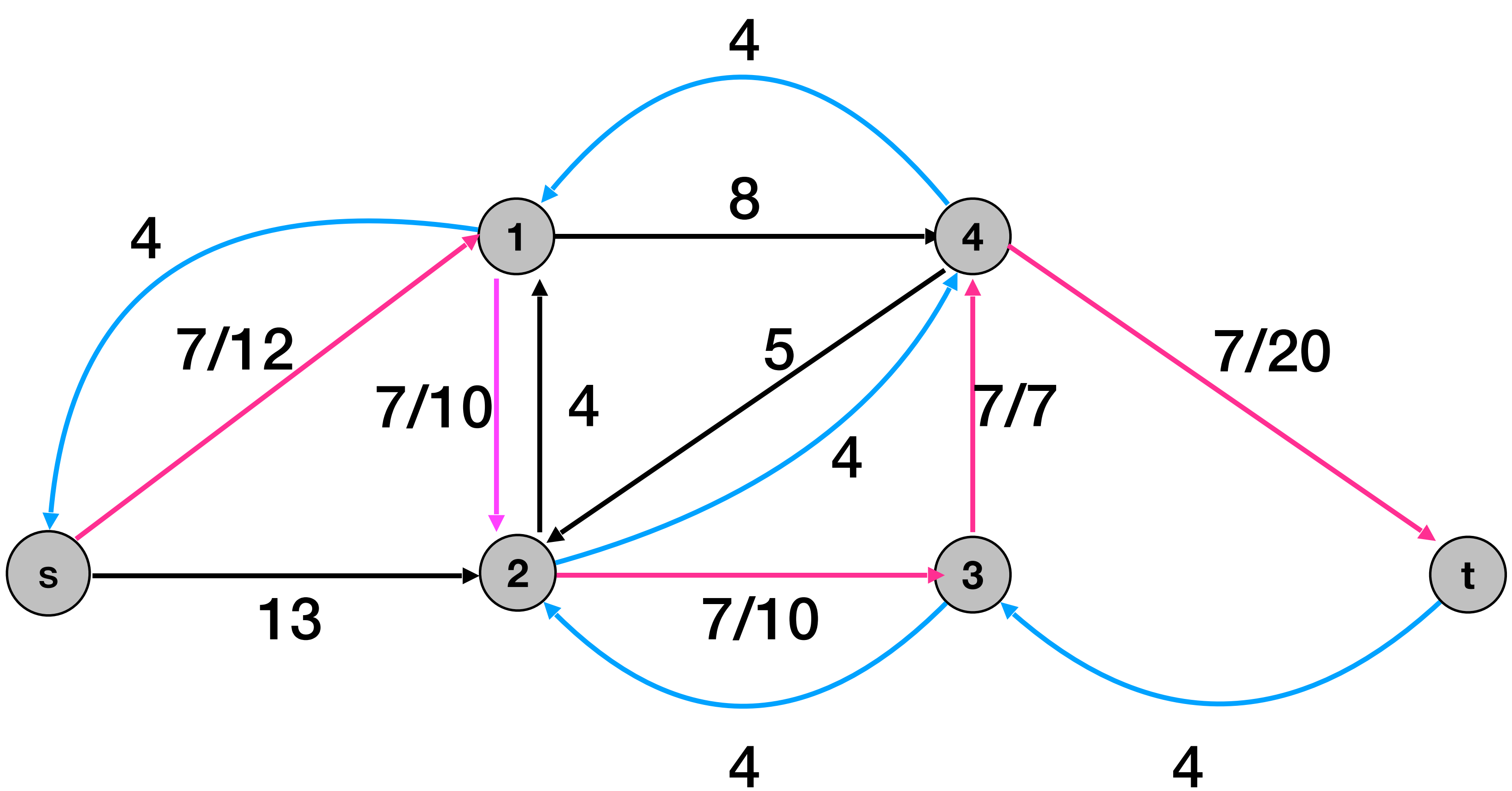
Ford-Fulkerson Algorithm



Residual Graph Gr

Use BFS to find an path P from s to t on Gr

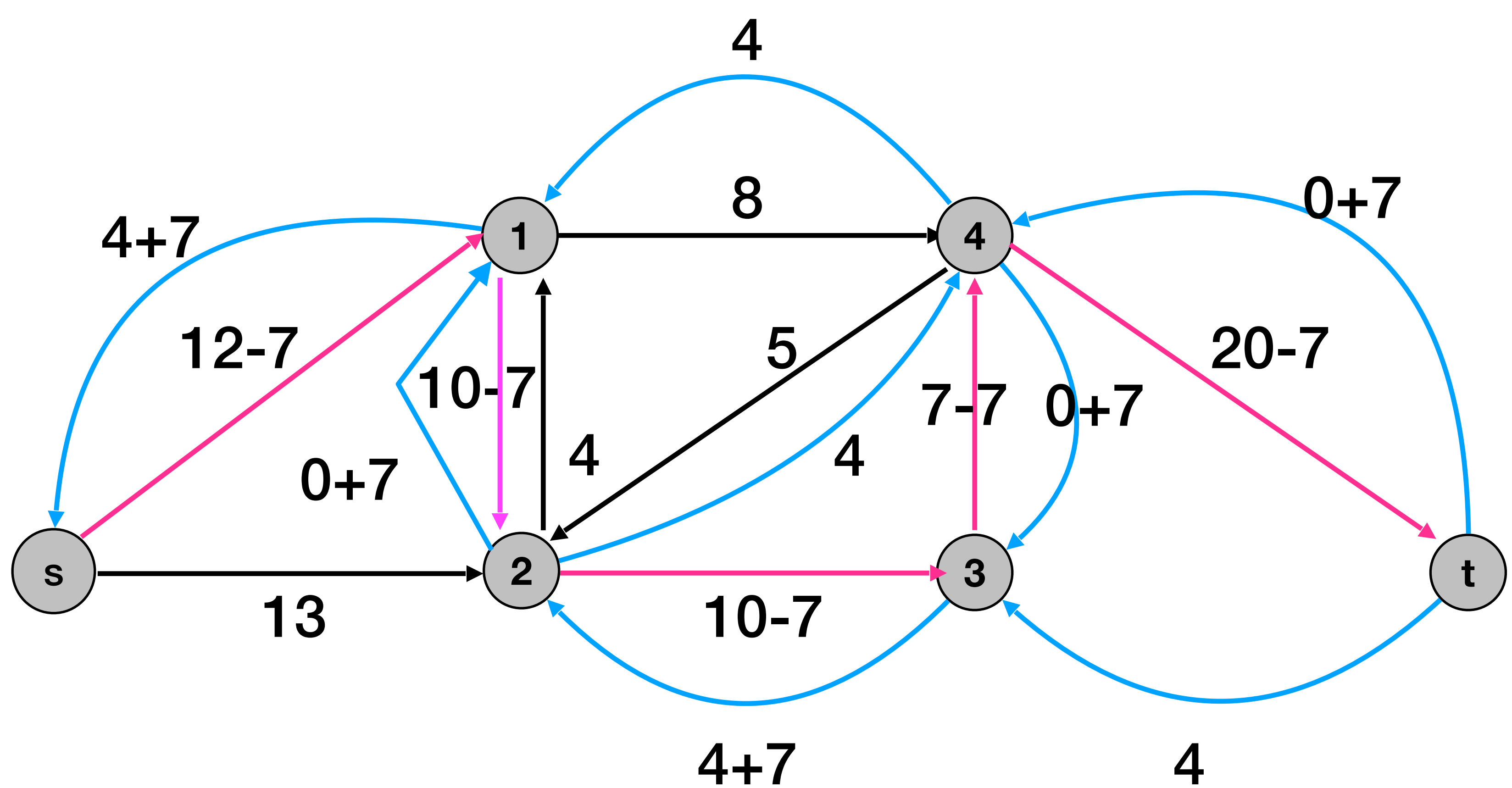
Ford-Fulkerson Algorithm



Residual Graph Gr

`Augment the flow on P of Gr`

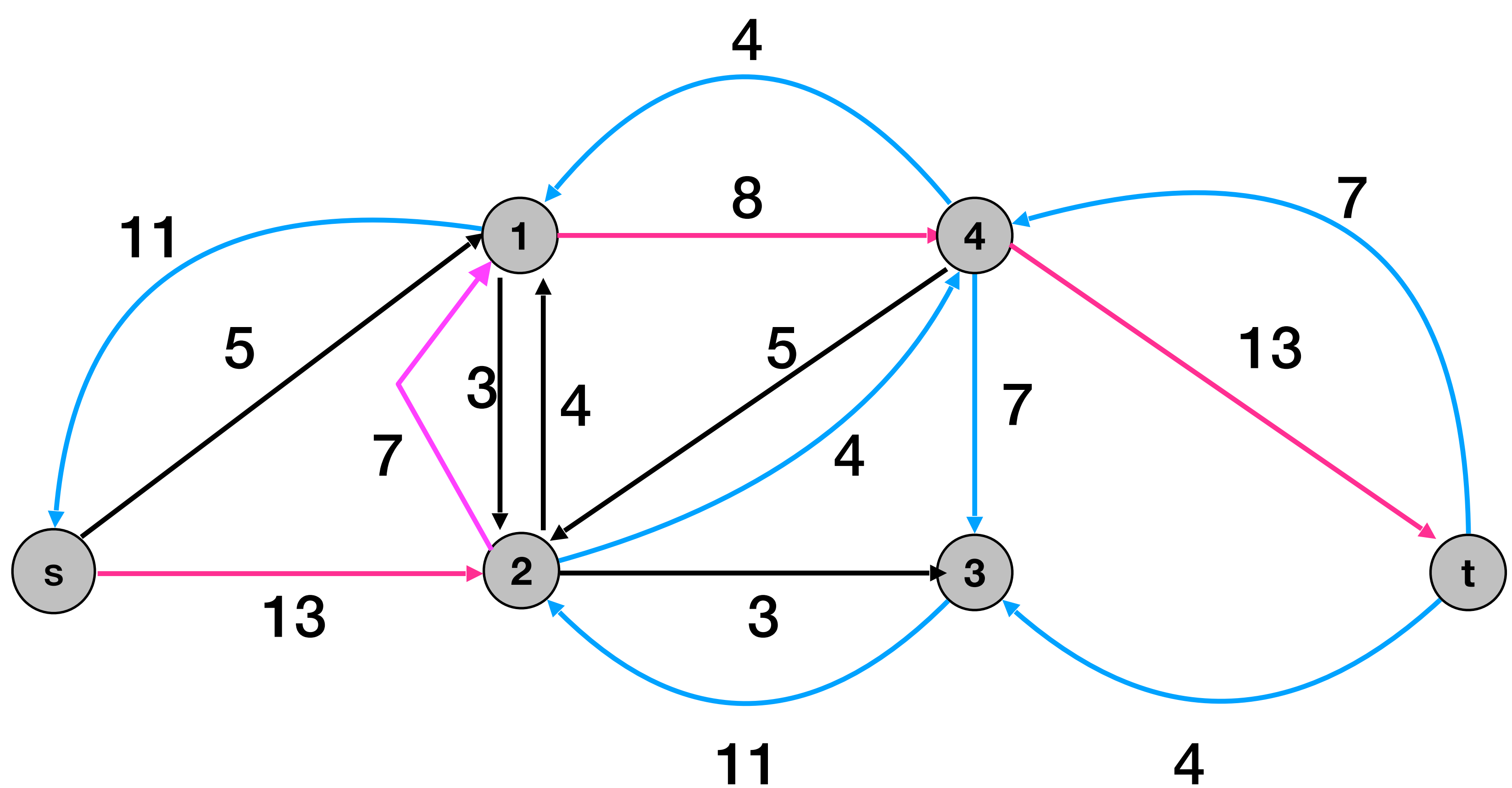
Ford-Fulkerson Algorithm



Residual Graph Gr

Update Gr

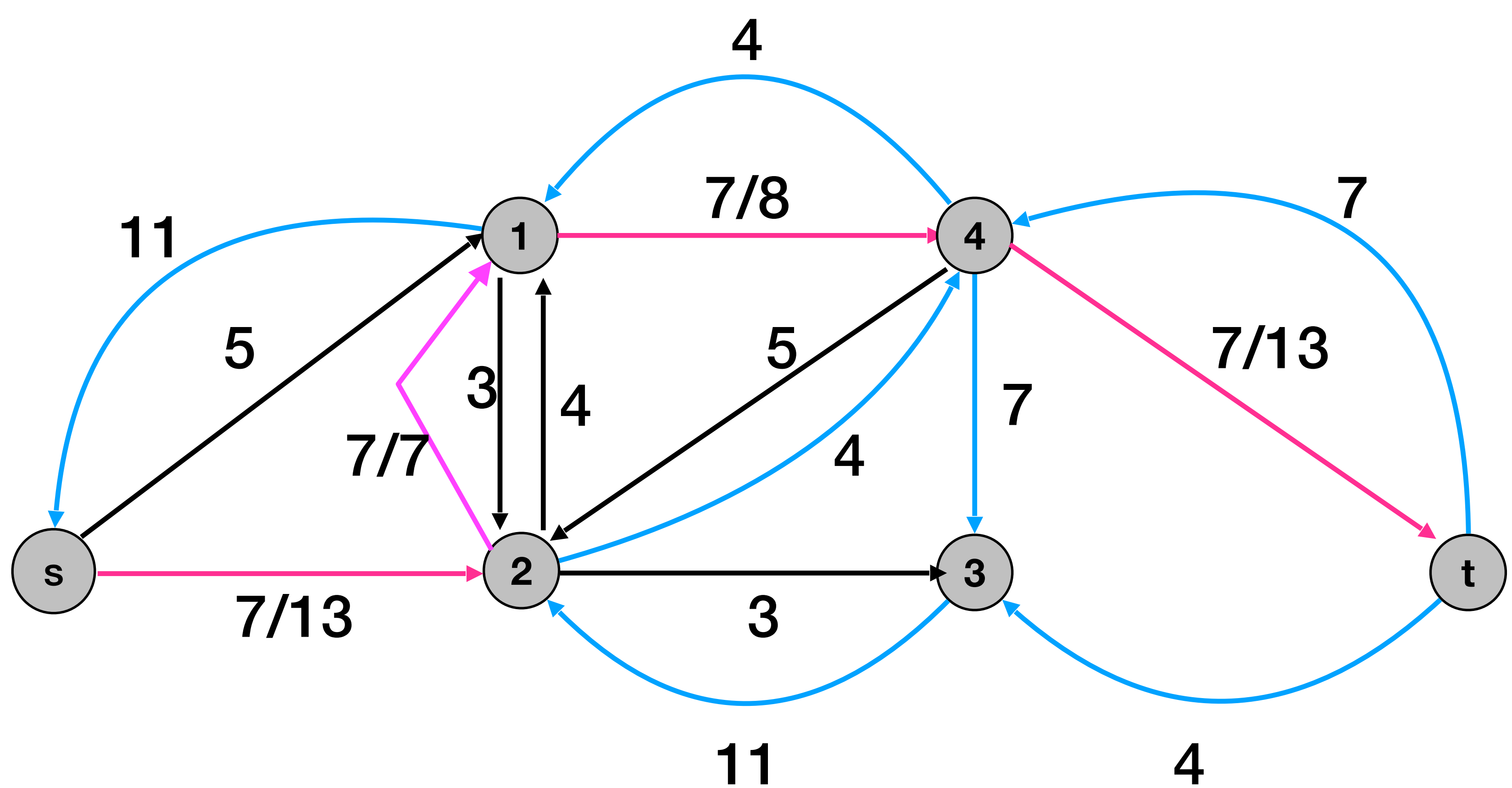
Ford-Fulkerson Algorithm



Residual Graph Gr

`Use BFS to find an path P from s to t on Gr`

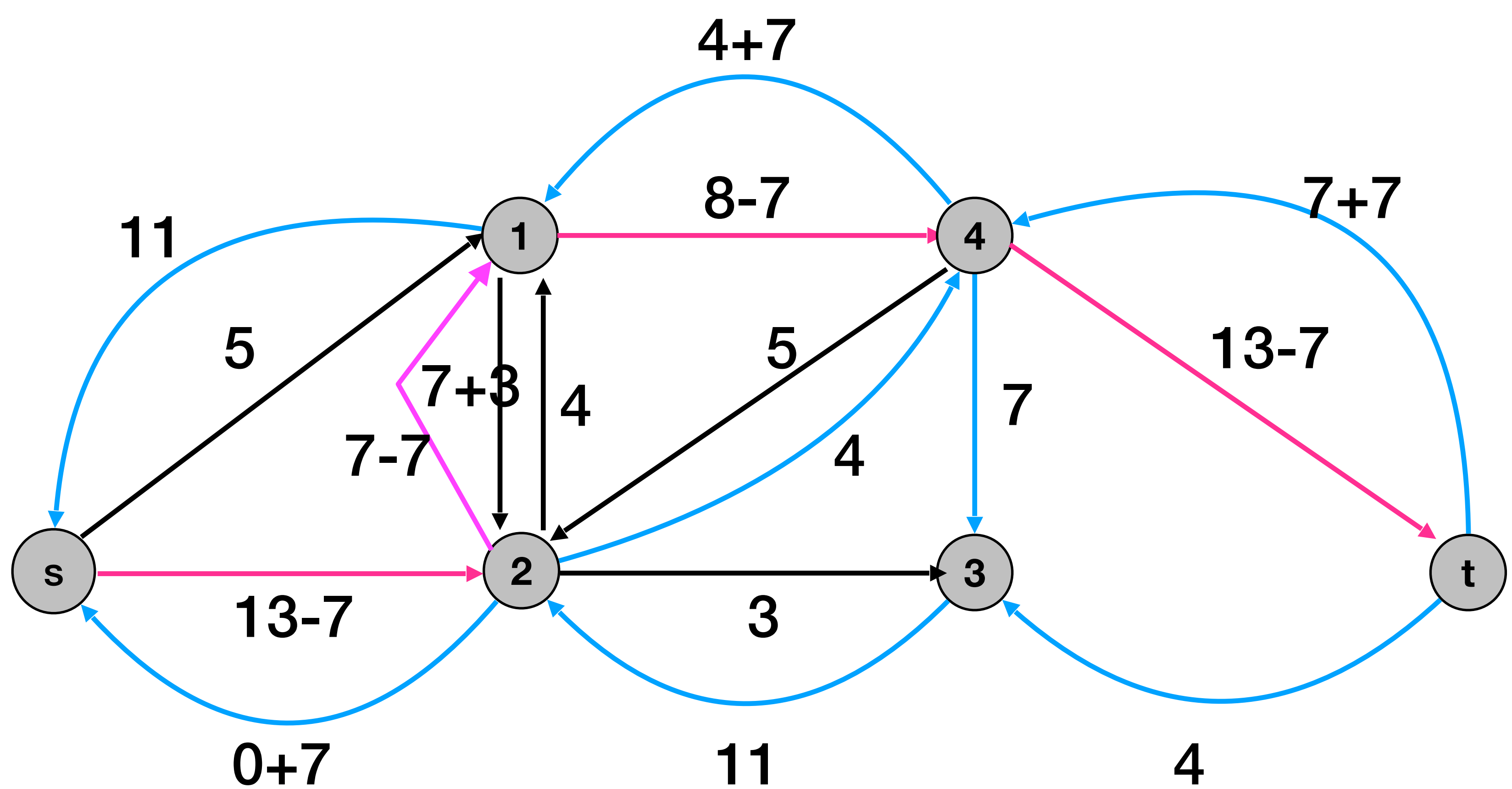
Ford-Fulkerson Algorithm



Residual Graph Gr

`Augment the flow on P of Gr`

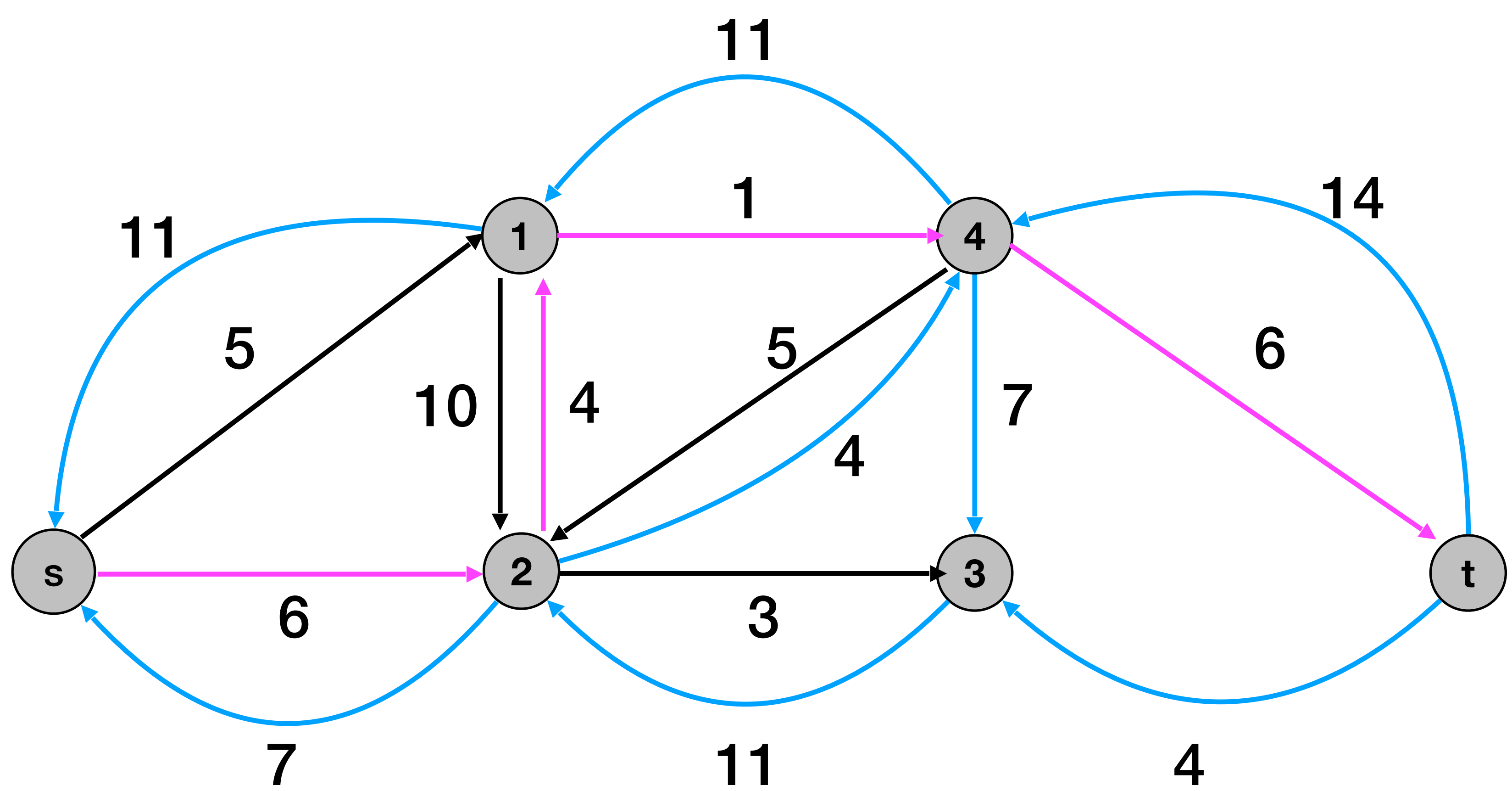
Ford-Fulkerson Algorithm



Residual Graph Gr

Update Gr

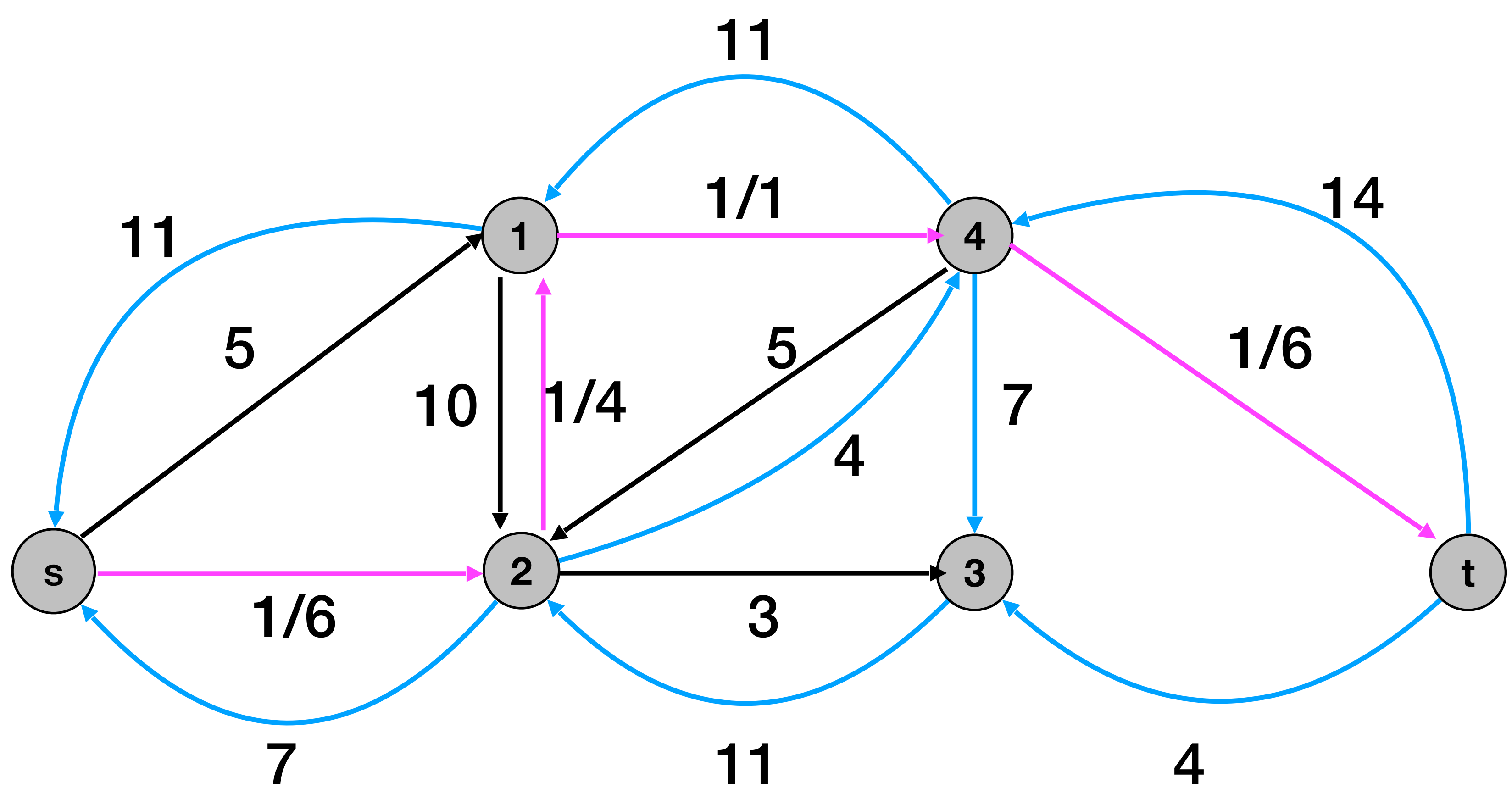
Ford-Fulkerson Algorithm



Residual Graph Gr

`Use BFS to find an path P from s to t on Gr`

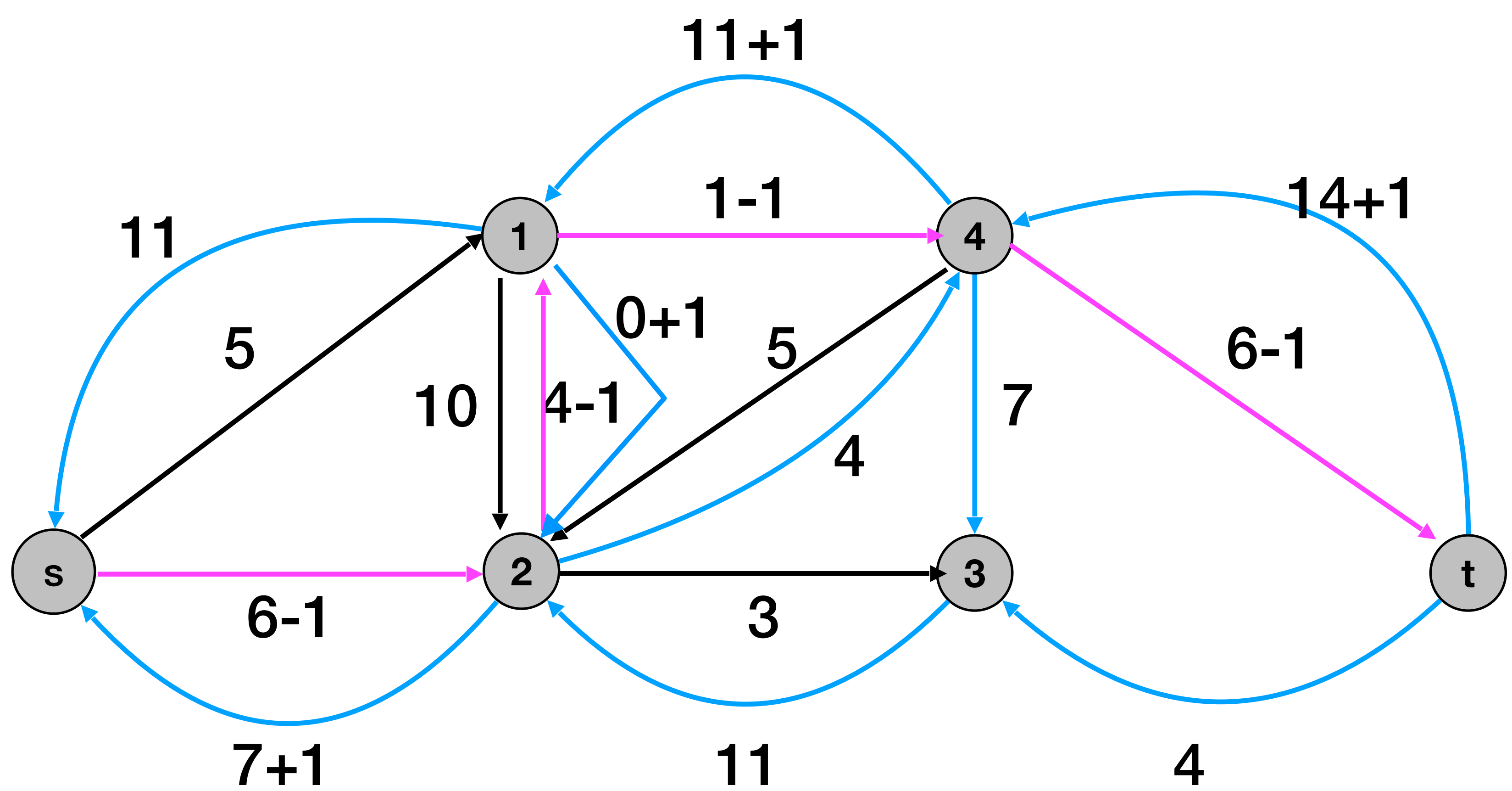
Ford-Fulkerson Algorithm



Residual Graph Gr

`Augment the flow on P of Gr`

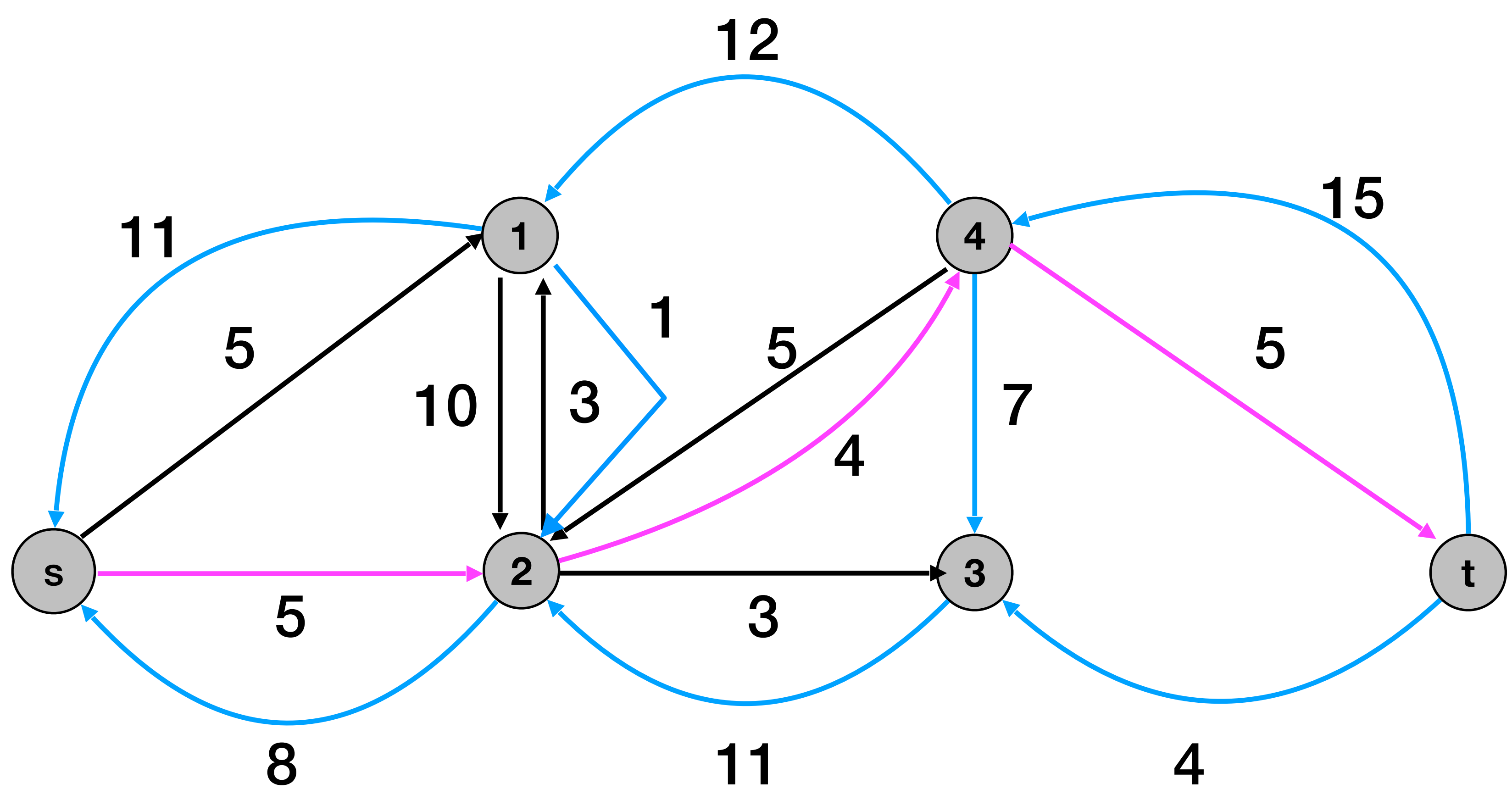
Ford-Fulkerson Algorithm



Residual Graph Gr

Update Gr

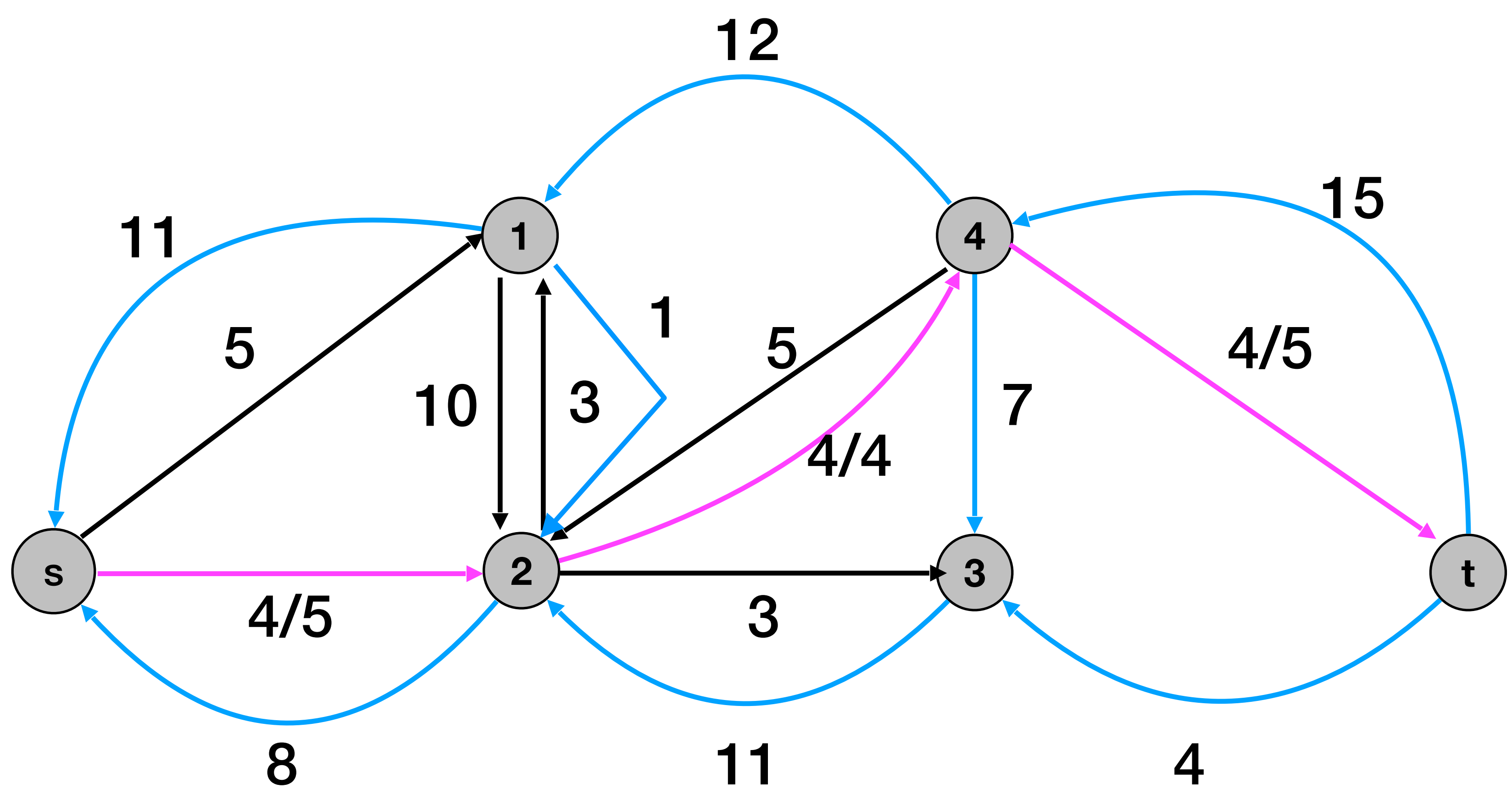
Ford-Fulkerson Algorithm



Residual Graph Gr

Use BFS to find an path P from s to t on Gr

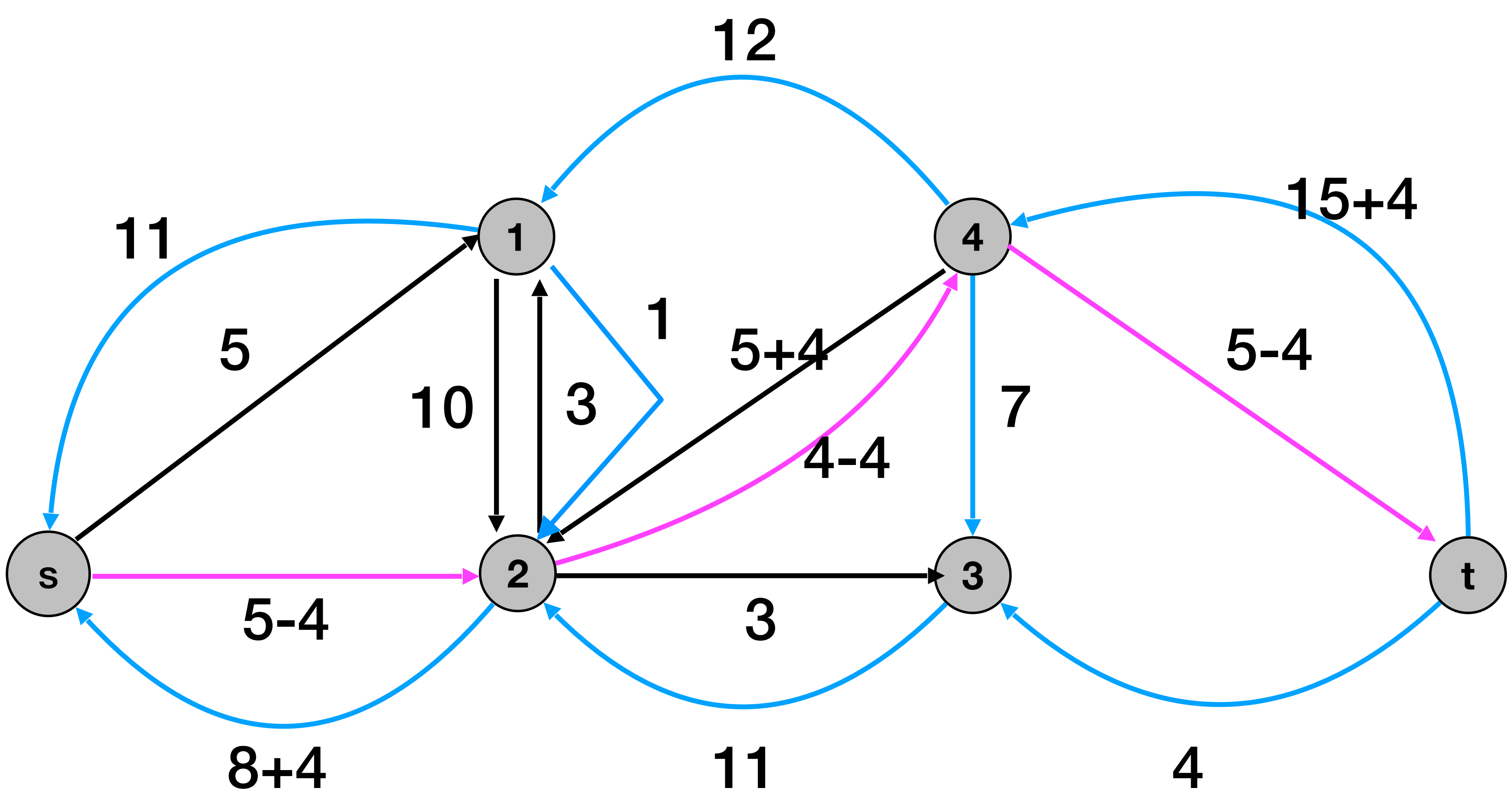
Ford-Fulkerson Algorithm



Residual Graph Gr

Augment the flow on P of Gr

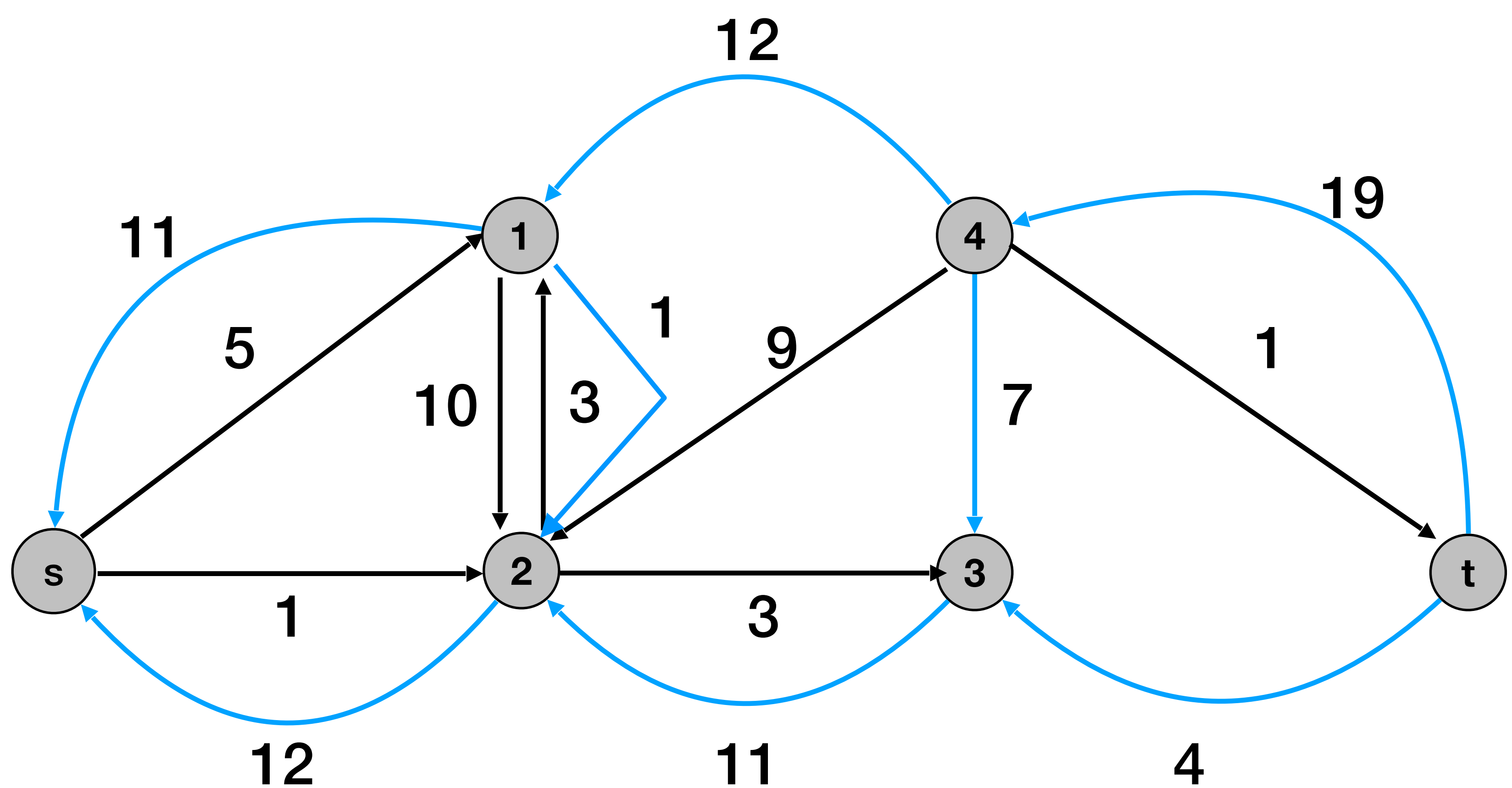
Ford-Fulkerson Algorithm



Residual Graph Gr

Update Gr

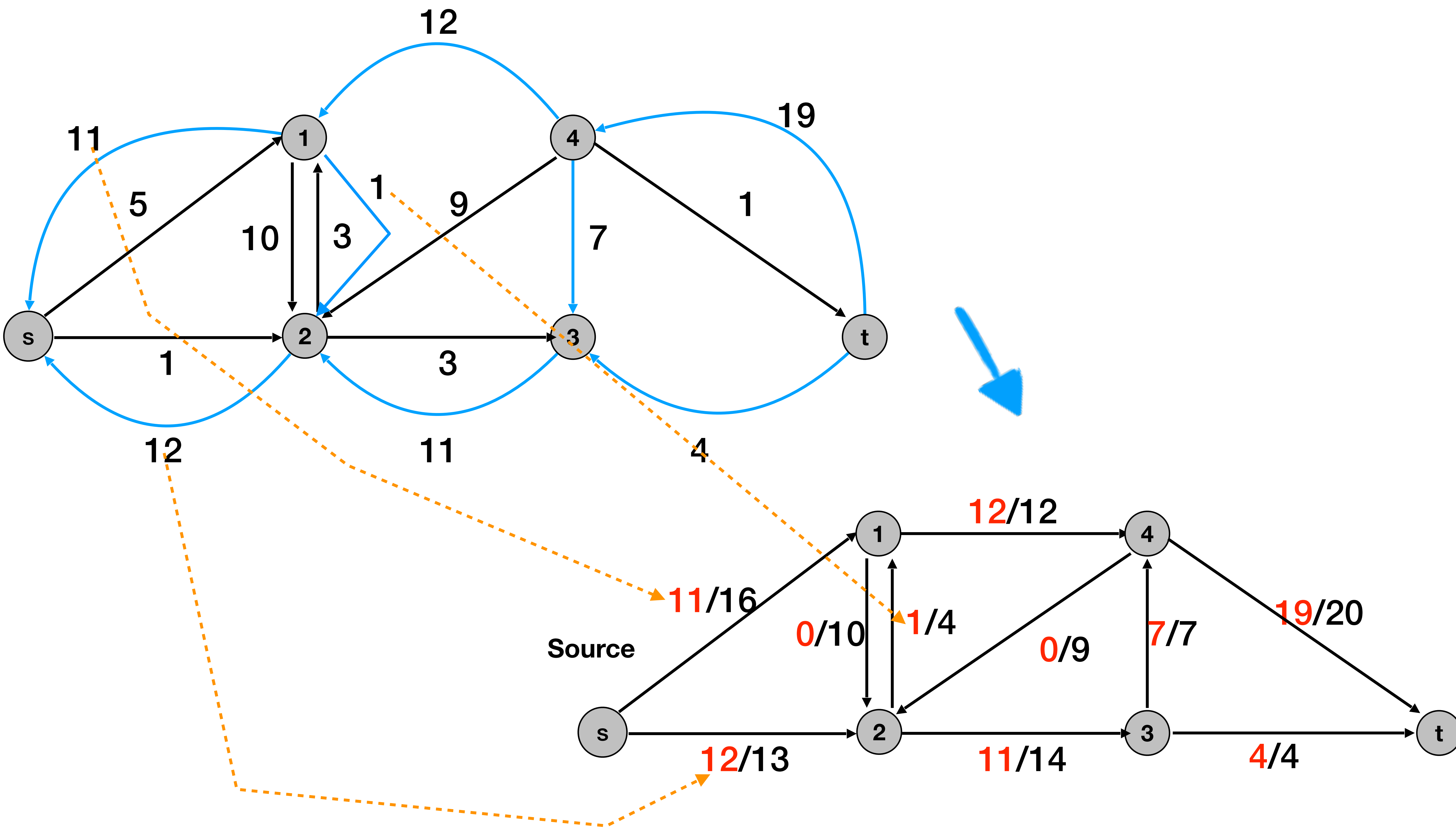
Ford-Fulkerson Algorithm



Residual Graph Gr

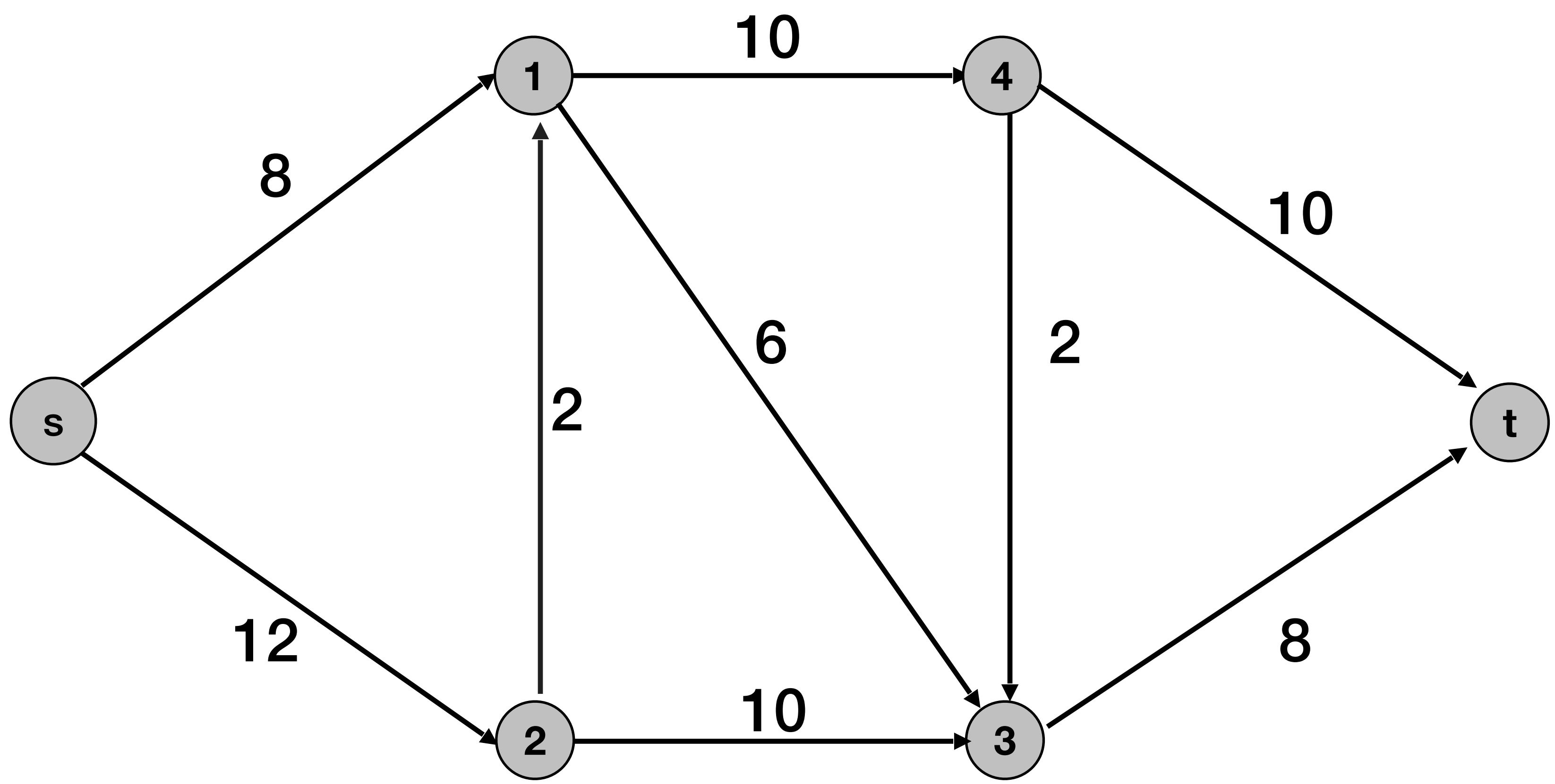
`There are no path from s to t in Gr: We are done.`

Ford-Fulkerson Algorithm

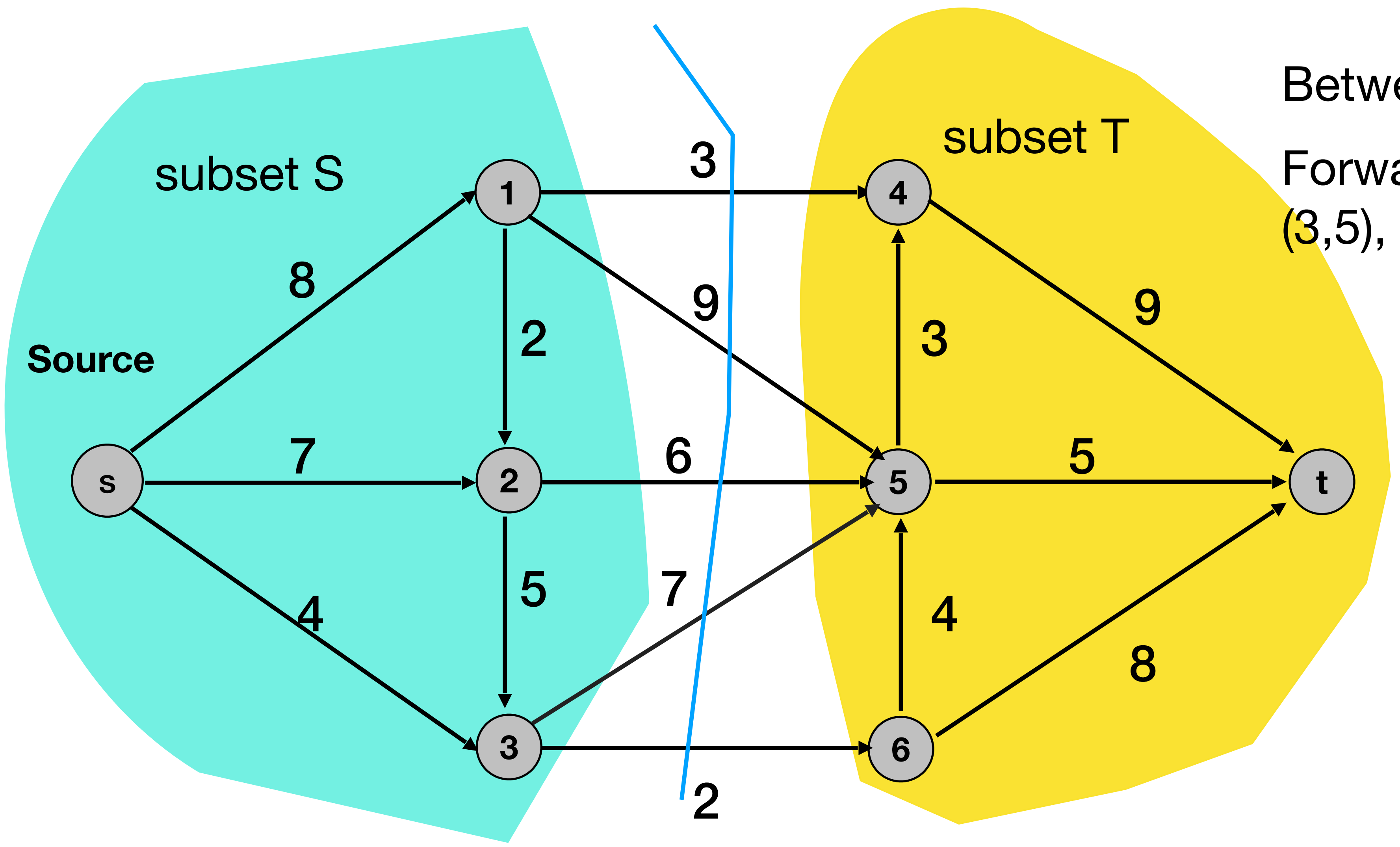


Back to the original network

Practice



Min Cut



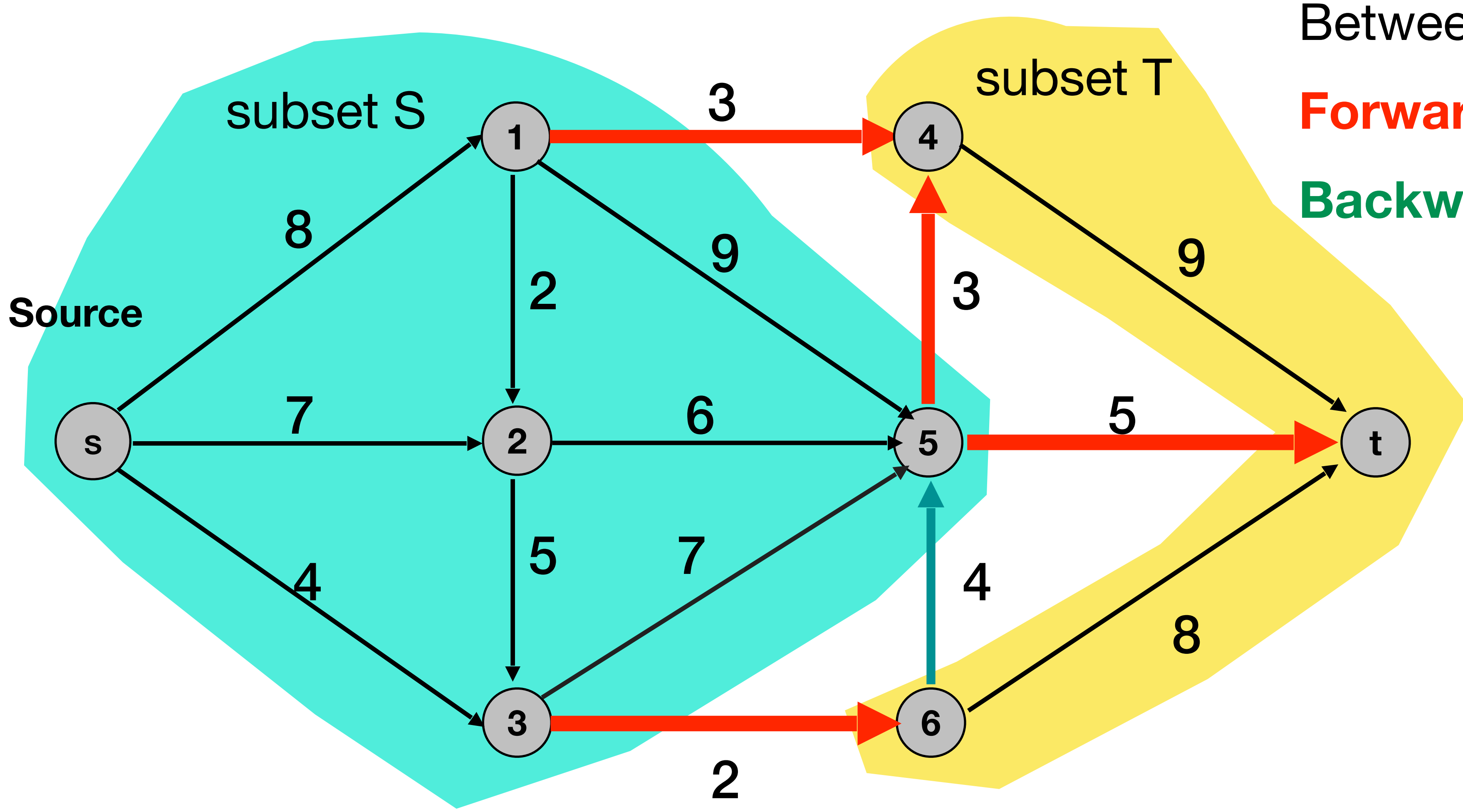
Between two subsets S and T,

Forward edges: {(1,4), (1,5), (2,5), (3,5), (3,6)}

A Cut is a node partition (S, T) such that s is in S and t is in T.

- e.g., S={s, 1, 2, 3} T = {4, 5, 6, t}
- Capacity of cut (S,T) is equal to the sum of capacities of forward edges between S and T.
- e.g., Capacity(S,T) = 3+ 9+ 6+ 7+ 2 = 27

Min Cut



Between two subsets S and T,

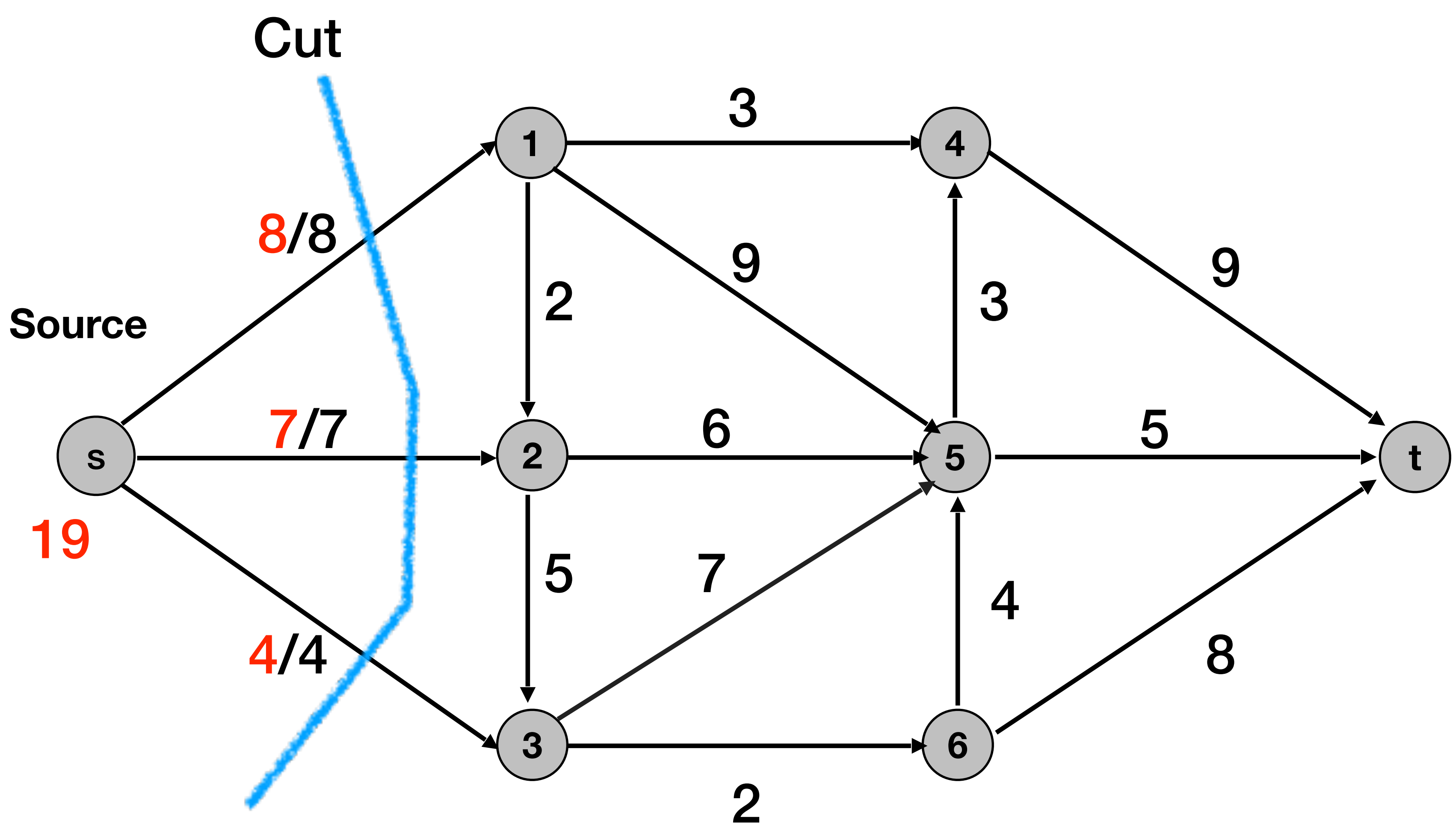
Forward edges: {(1,4),(5,4),(5,t),(3,6)}

Backward edges: (6,5)

A min cut is a cut among all cuts with the **minimum** capacity.

- e.g., S={s, 1, 2, 3, 5}   T = {4, 6, t}
- e.g., Capacity(S,T) = 3+ 3+ 5+ 2 = 13

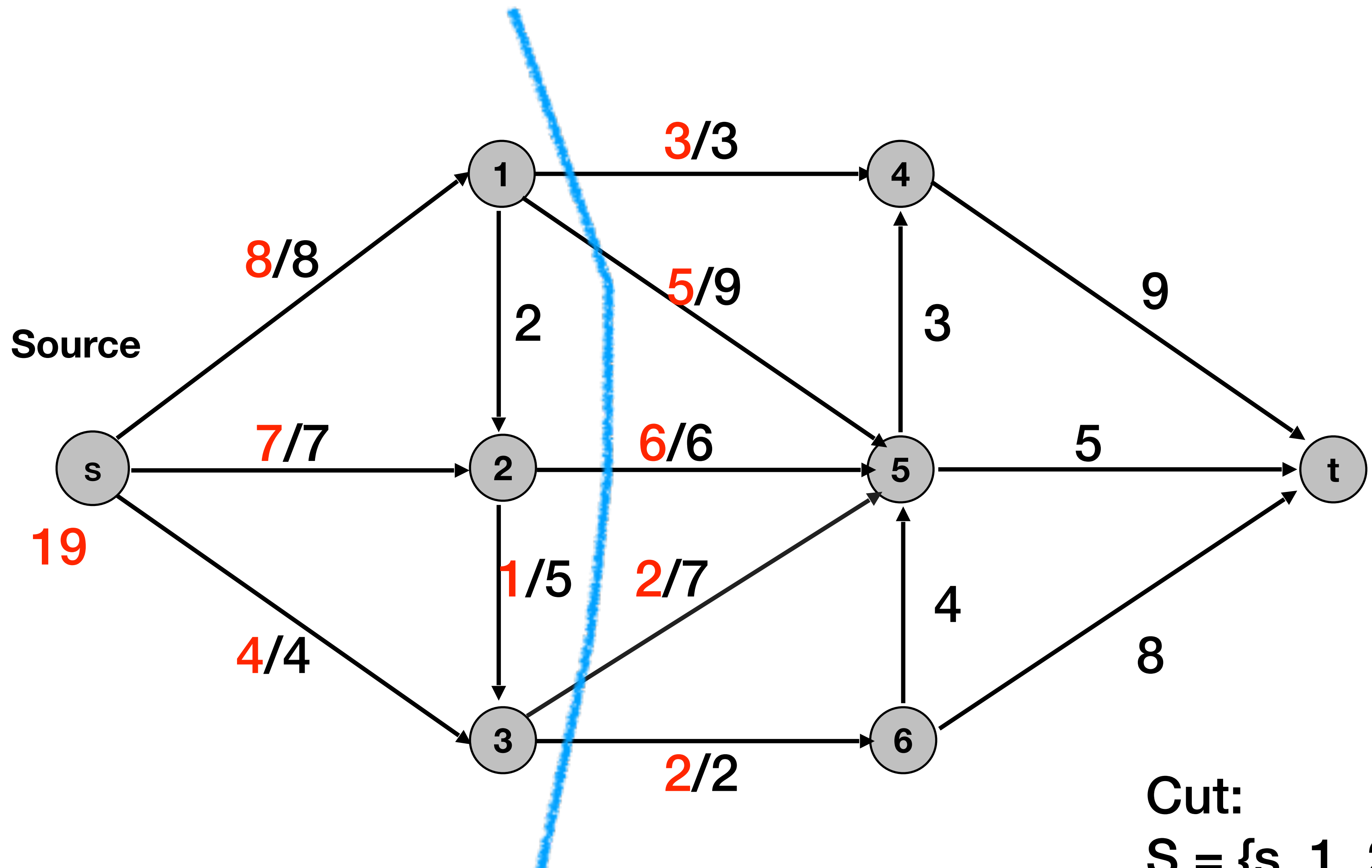
Max flow Min Cut



Factory as city s transports flow (freight) 19 to city t one day

Observation: Let f be a flow, and let (S, T) be any s-t cut. Then, the flow sent across the cut is at most the capacity of this cut.

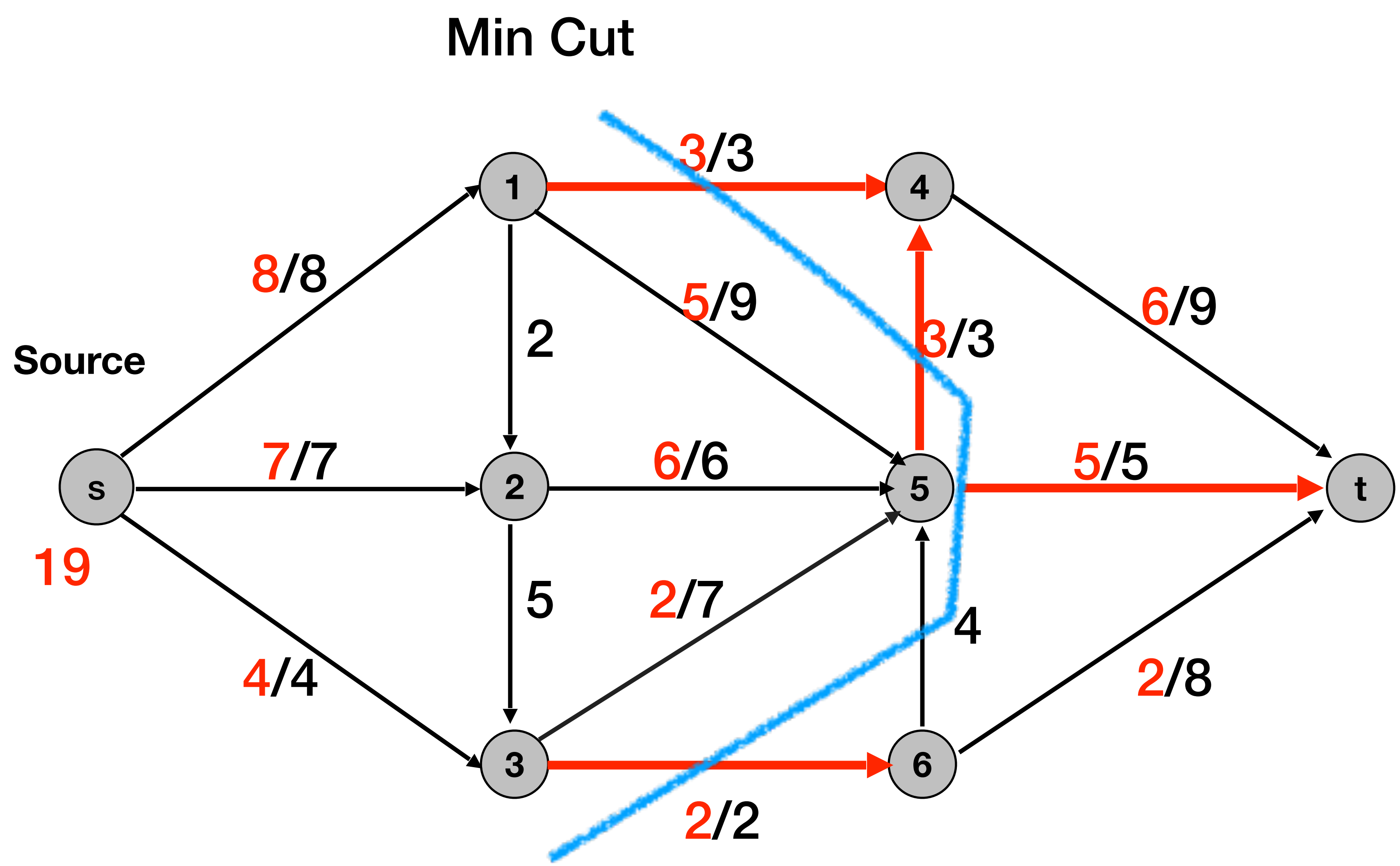
Max flow Min Cut



Cut:
S = {s, 1, 2, 3}
T = {4, 5, 6, t}
Capacity (S, T) = 27

Observation: Let f be a flow, and let (S, T) be any s-t cut. Then, the flow sent across the cut is at most the capacity of this cut.

Max flow Min Cut

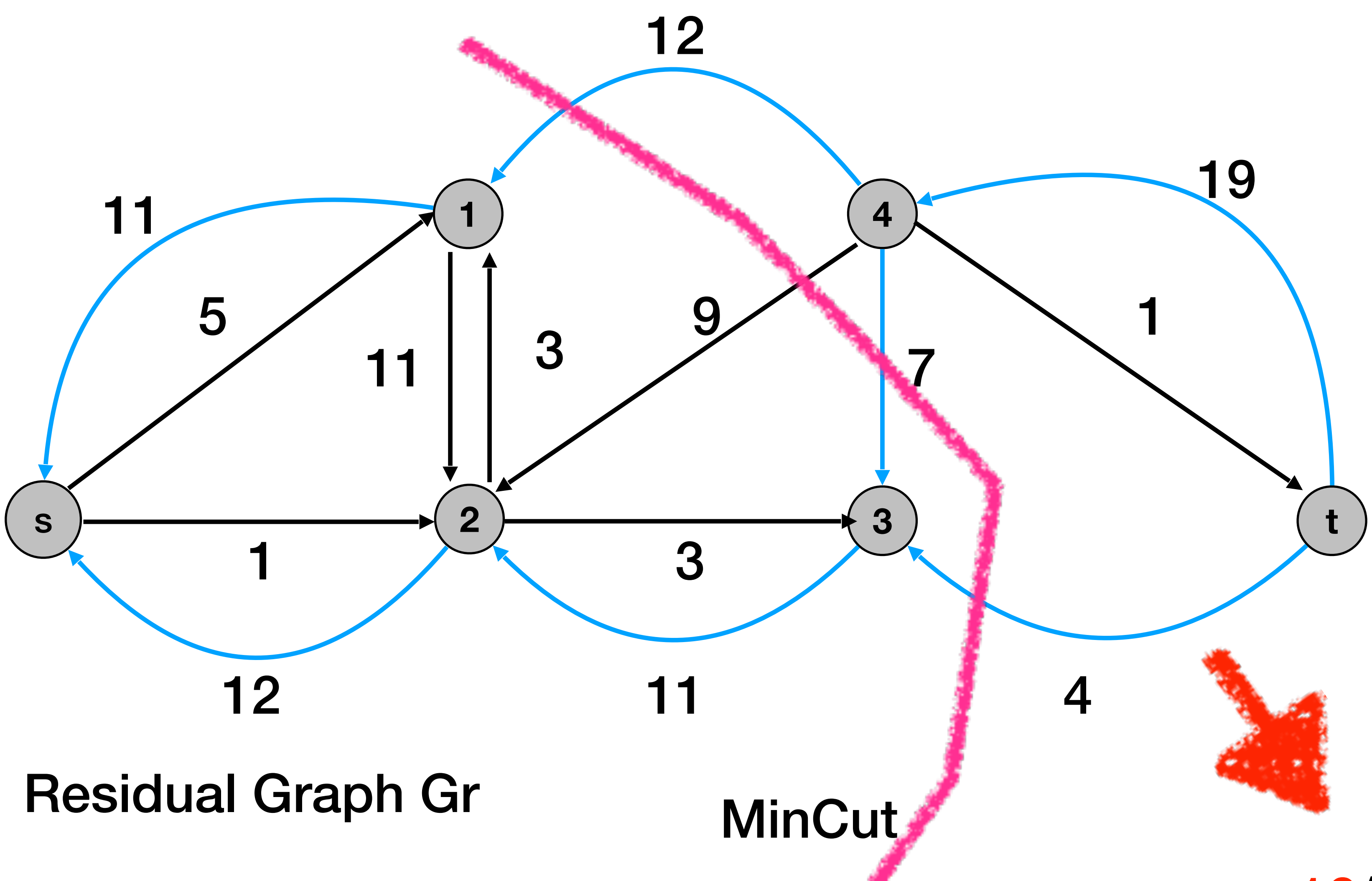


Flow 13 arrives at city t

Min Cut:
S={s, 1, 2, 3, 5}
T={4, 6, t}
Capacity (S, T) = 13

Max-flow min-cut theorem. (Ford-Fulkerson, 1956):
In any network, the value of max flow equals capacity of min cut.

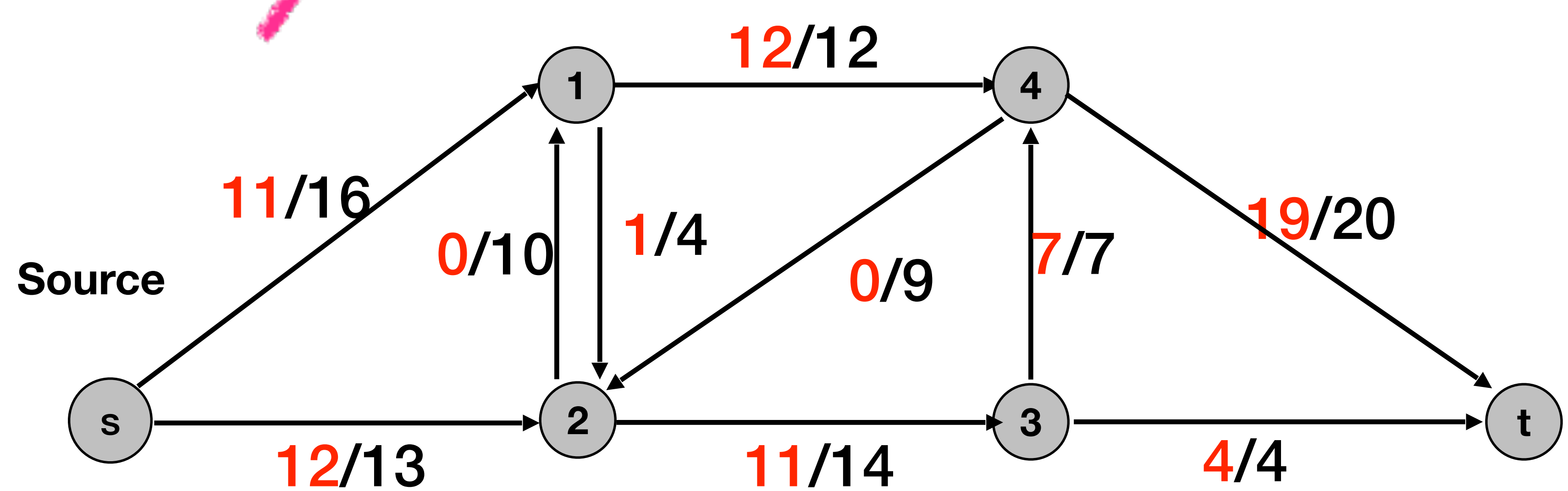
Ford-Fulkerson Algorithm



Residual Graph Gr

MinCut:
S={s, 1, 2, 3}
T={4,t}

Observation:
No path exists from any v in S to any u in T



Original Graph G

Application - cargo transportation