Algorithms and Data Structures 2
CS 1501

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
  - Homework 11: Friday 12/9 @ 11:59 pm
  - Lab 12: Monday 12/12
  - Homework 12: Monday 12/19
  - Assignment 5 is now for extra credit ONLY
  - We have 4 programming assignments
    - the lowest is dropped
    - each worth 13.3%
  - Assignment 3: Friday 12/16 @ 11:59 pm
  - Assignment 4: Friday 12/16 @ 11:59 pm

Bonus Opportunities

- Bonus Lab due on 12/19
- Bonus Homework due on 12/19
- Bonus Assignment due on 12/19
- 1 bonus point for entire class when OMETs response rate >= 80%
  - Currently at ~32% for MW and 28% for TuTh
  - Deadline is Sunday 12/11

Final Exam

- Same format as midterm
- Non-cumulative
- Date, time and location on PeopleSoft
  - MW Section: Monday 12/12 8-9:50 am (coffee served)
  - TuTh Section: Thursday 12/15 12:00-1:50 pm
- Same classroom as lectures
- Study guide and practice test to be posted soon

Previous Lecture

- Network Flow Problem
  - Ford Fulkerson's Framework
    - augmenting paths on the residual graph

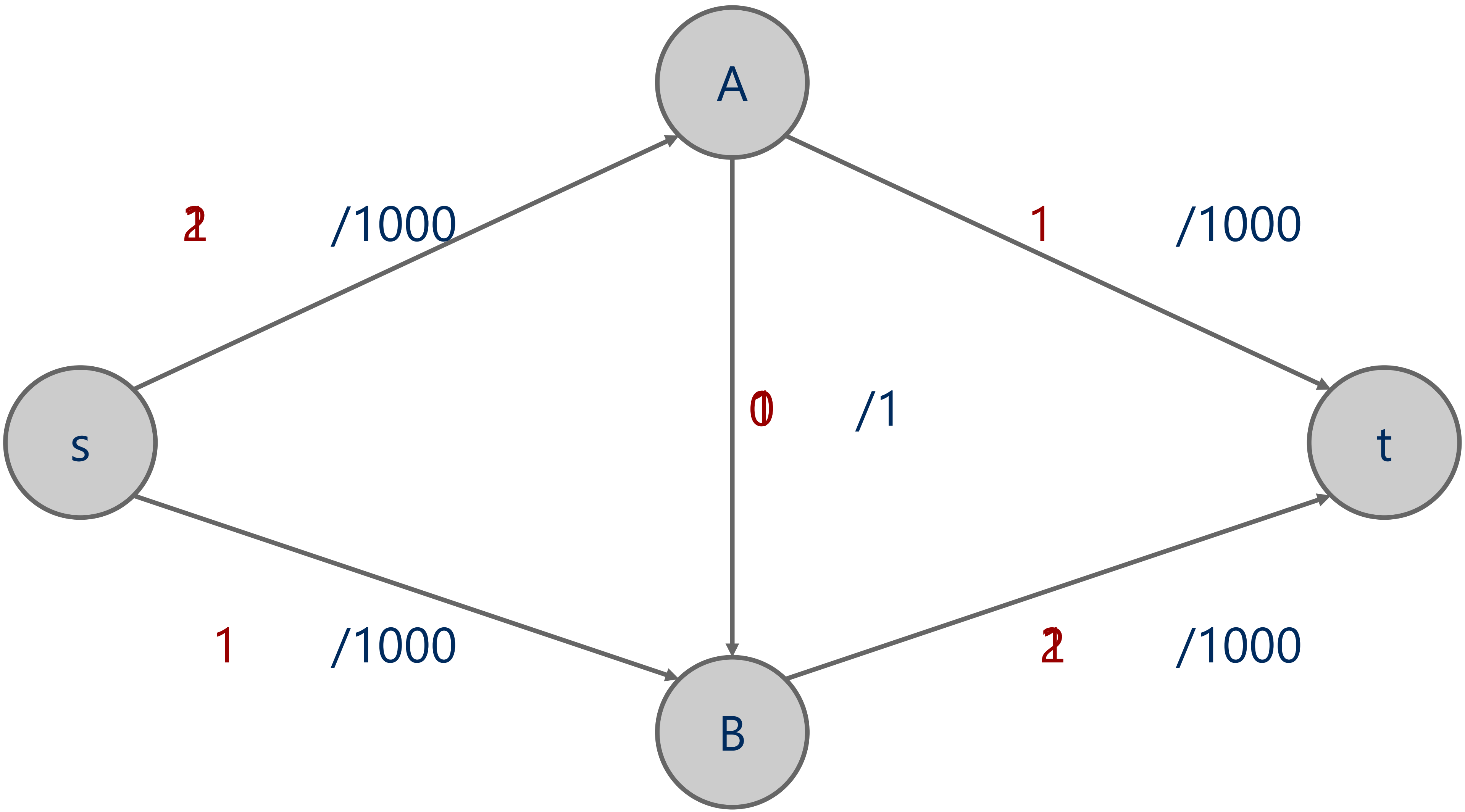
This Lecture

- Two specific ways of finding augmenting paths
  - BFS (Edmonds-Karp)
  - Priority First Search (PFS)
- The minimum-cut problem

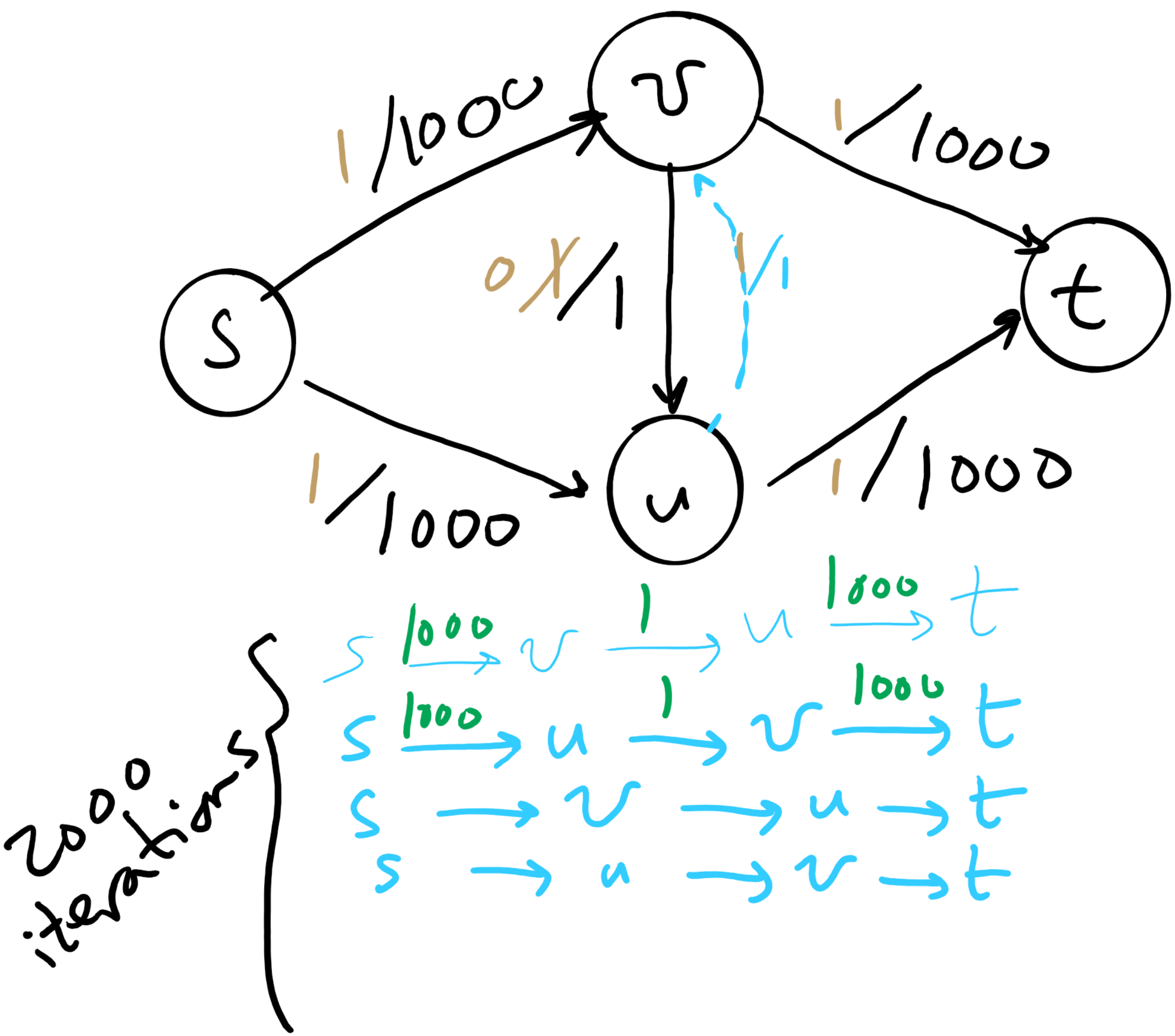
Problem of the Day: Finding Bottlenecks

- Let’s assume that we want to send a large file from point A to point B over a computer network as fast as possible over multiple network links if needed
- Input:
  - A computer network
    - Network nodes and links
    - Links are labeled by link capacity in Mbps
  - Starting node and destination node
- Output:
  - The maximum network speed possible for sending a file from source to destination

Another example



Worst-case runtime of Ford-Fulkerson



Worst-case Runtime of Ford-Fulkerson

$$\Theta\left(|f| * (e+v)\right)$$

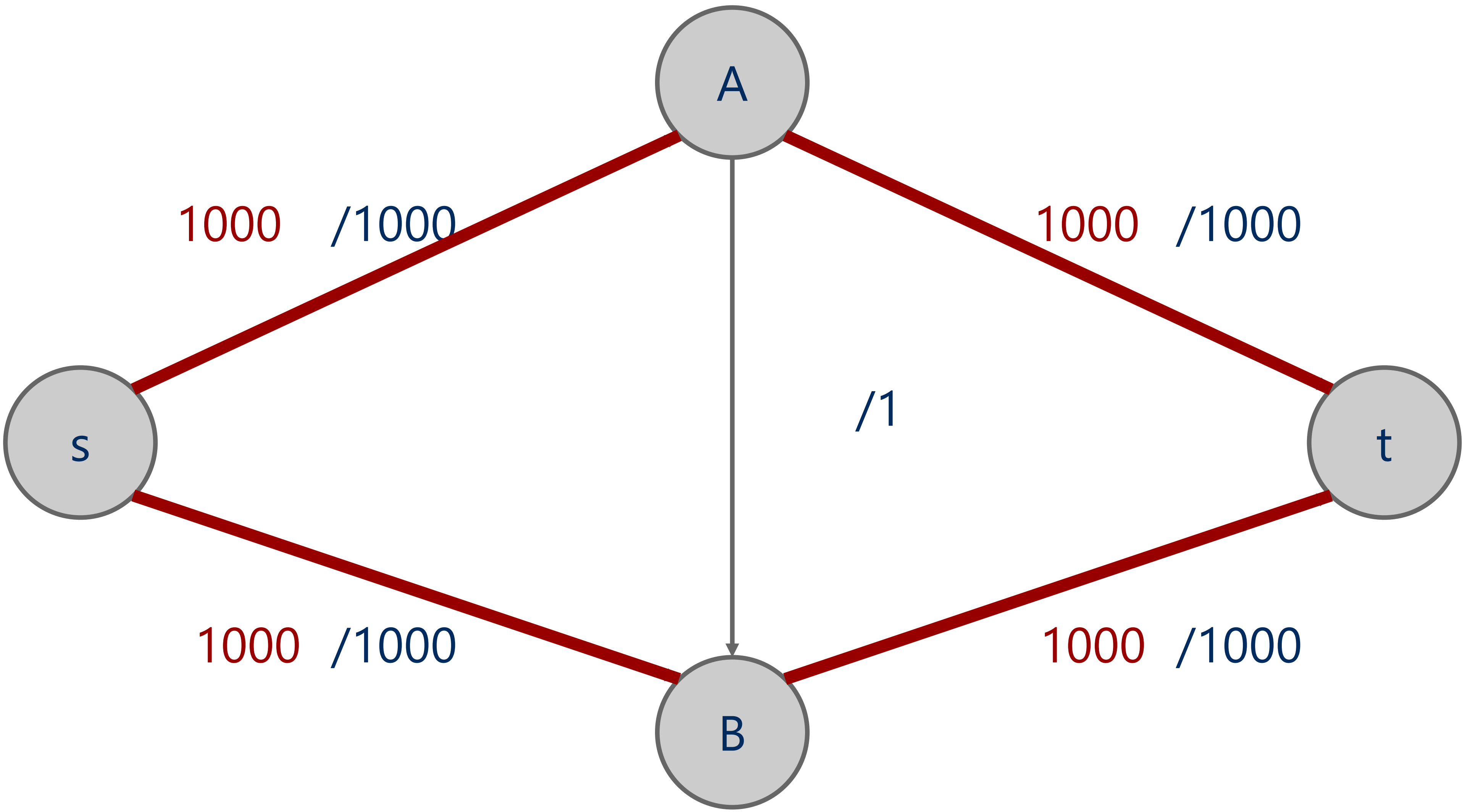
Max. Flow Value

time for finding an augmenting Path

Edmonds Karp

- How the augmenting path is chosen affects the performance of the search for max flow
- Edmonds and Karp proposed a shortest path heuristic for Ford Fulkerson
  - Use BFS to find augmenting paths

Another example



But our flow graph is weighted...

- Edmonds-Karp only uses BFS
  - Used to find spanning trees and shortest paths for *unweighted* graphs
  - Why do we not use some measure of priority to find augmenting paths?

But our flow graph is weighted...

- Edmonds-Karp only uses BFS
  - Used to find spanning trees and shortest paths for *unweighted* graphs
  - Why do we not use some measure of priority to find augmenting paths?

Maximum Capacity Path

- Proposed by Edmonds and Karp
- implemented by modifying Dijkstra's shortest paths algorithm
  - Define **flow[v]** as the maximum amount of flow from s → v along a *single path*
  - Each iteration set curr as the unmarked vertex with the largest flow[]
  - For each neighbor w of curr,
    - if more flow can be pumped to w through curr, update flow[w]
    - if **min(flow[curr], residual capacity of (curr, w))** > flow[w]
      - update flow[w] and parent[w] to be curr

Flow edge implementation

- For each edge, we need to store:
  - Start point, the from vertex
  - End point, the to vertex
  - Capacity
  - Flow
  - Residual capacities
    - For forwards and backwards edges

FlowEdge.java

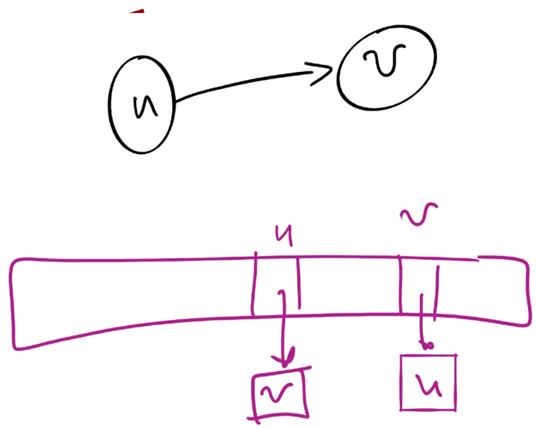
```
public class FlowEdge {
   private final int v;                 // from
   private final int w;                 // to
   private final double capacity;    // capacity
   private double flow;                 // flow
      …
      public double residualCapacityTo(int vertex) {
       if       (vertex == v) return flow;
       else if (vertex == w) return capacity - flow;
       else throw new
        IllegalArgumentException("Illegal endpoint");
   }
   …
}
```

BFS search for an augmenting path (pseudocode)

```
edgeTo = [|V|]
marked = [|V|]
Queue q
q.enqueue(s)
marked[s] = true
while !q.isEmpty():
    v = q.dequeue()
    for each (v, w) in AdjList[v]:
        if residualCapacity(v, w) > 0:
            if !marked[w]:
                edgeTo[w] = v;
                marked[w] = true;
                q.enqueue(w);
```

Each FlowEdge object is stored in the adjacency list twice:

Once for its forward edge
Once for its backward edge



Value of maxflow

- Add up the flow increments in each iteration of Ford-Fulkerson
- Add up the edge **flows** out of source
- Add up the edge **flows** of the out of source

Follow-up Problem

- So, now we found the bottleneck *value*, but which edges define the found bottleneck?
  - *Why would you want to know those bottleneck edges?*

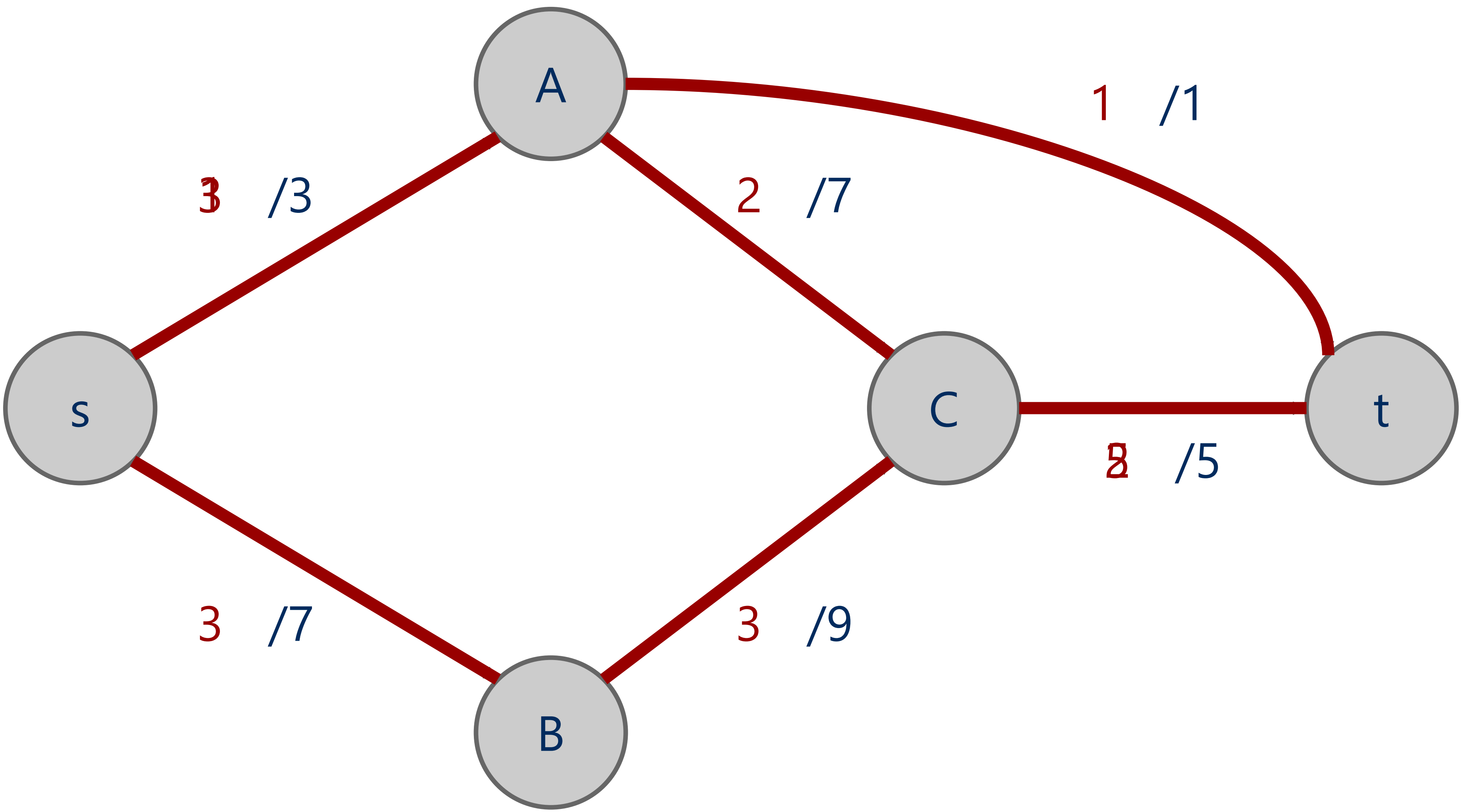
Let's separate the graph

- An st-cut on G is a set of edges in G that, if removed, will partition the vertices of G into two disjoint sets
  - One contains s
  - One contains t
- May be many st-cuts for a given graph
- Let's focus on finding the minimum st-cut
  - The st-cut with the smallest capacity
  - May not be unique

How do we find the min st-cut?

- We could examine residual graphs
    - Specifically, try and allocate flow in the graph until we get to a residual graph with no existing augmenting paths
        - A set of saturated edges will make a minimum st-cut

Min cut example



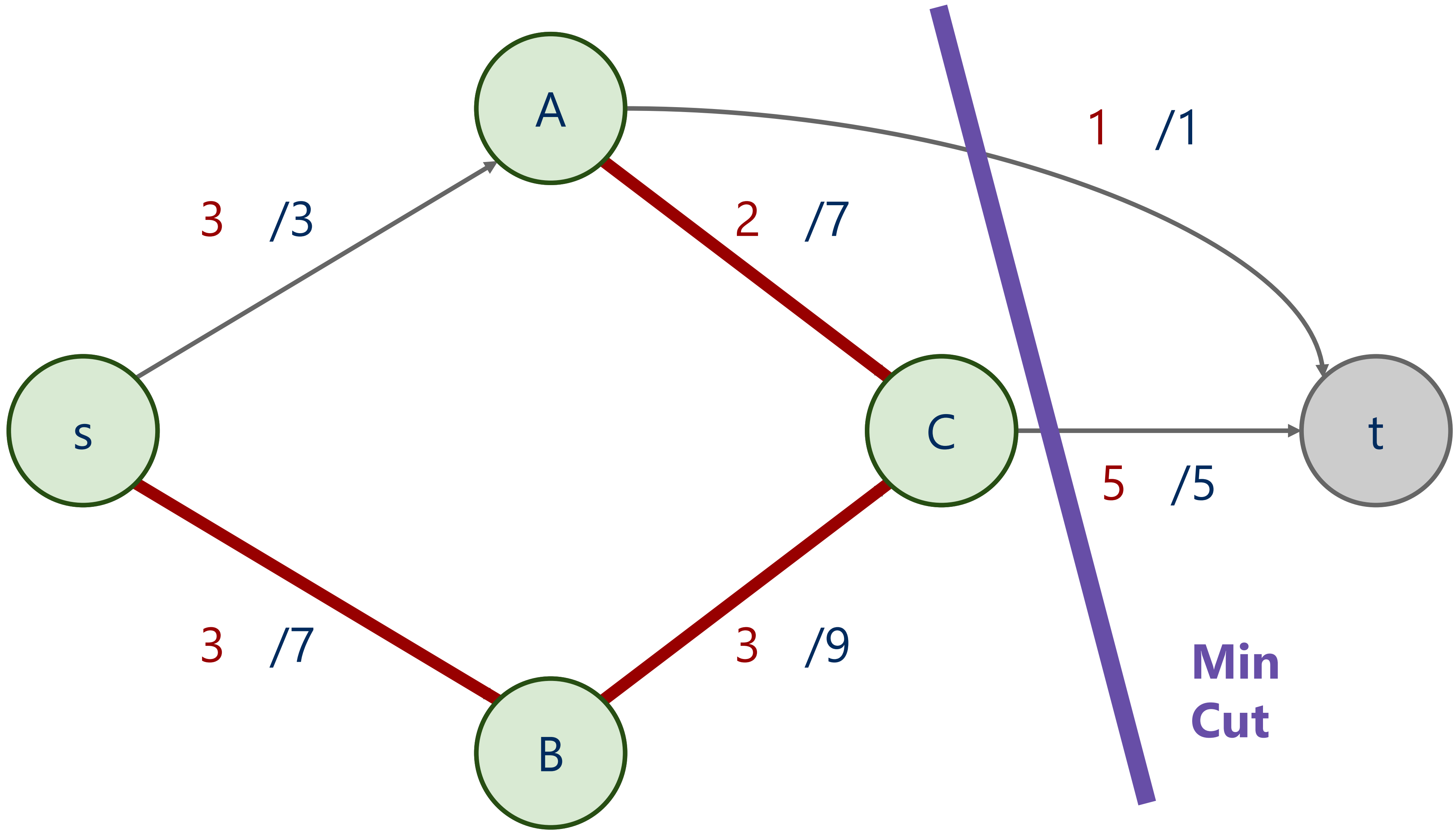
Max flow == min cut

- A special case of duality
  - I.e., you can look at an optimization problem from two angles
    - In this case to find the maximum flow or minimum cut
  - In general, dual problems do not have to have equal solutions
    - The differences in solutions to the two ways of looking at the problem is referred to as the *duality gap*
      - If the duality gap = 0, strong duality holds
        - Max flow/min cut uphold strong duality
      - If the duality gap > 0, weak duality holds

Determining a minimum st-cut

- First, run Ford Fulkerson to produce a residual graph with no further augmenting paths
- The last attempt to find an augmenting path will visit every vertex reachable from s
    - Edges with only one endpoint in this set comprise a minimum st-cut

Determining the min cut



Max flow / min cut on unweighted graphs

- Is it possible?
- How would we measure the Max flow / min cut?
- What would an algorithm to solve this problem look like?

Unweighted network flow