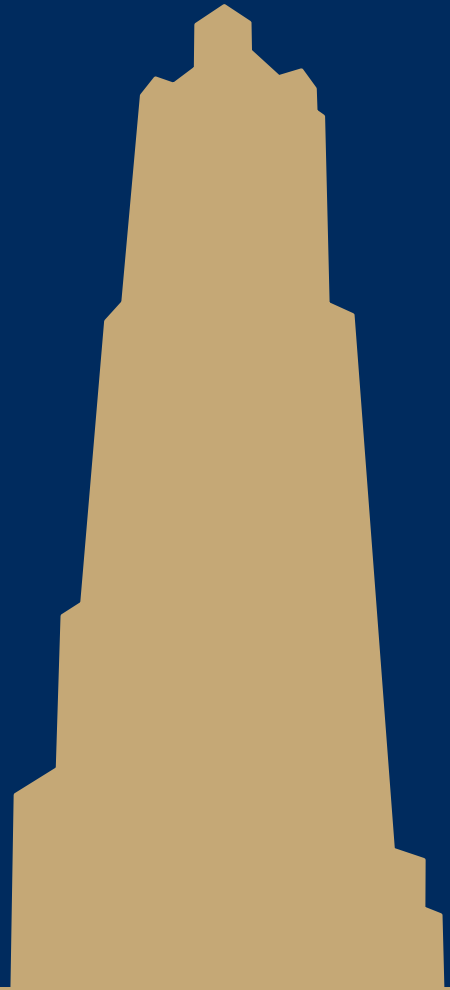


CS/COE 1501

www.cs.pitt.edu/~nlf4/cs1501/

Network Flow



Defining network flow

- Consider a directed, weighted graph $G(V, E)$
 - Weights are applied to edges to state their *capacity*
 - $c(u, w)$ is the capacity of edge (u, w)
 - if there is no edge from u to w , $c(u, w) = 0$
- Consider two vertices, a *source* s and a *sink* t
 - Let's determine the maximum flow that can run from s to t in the graph G

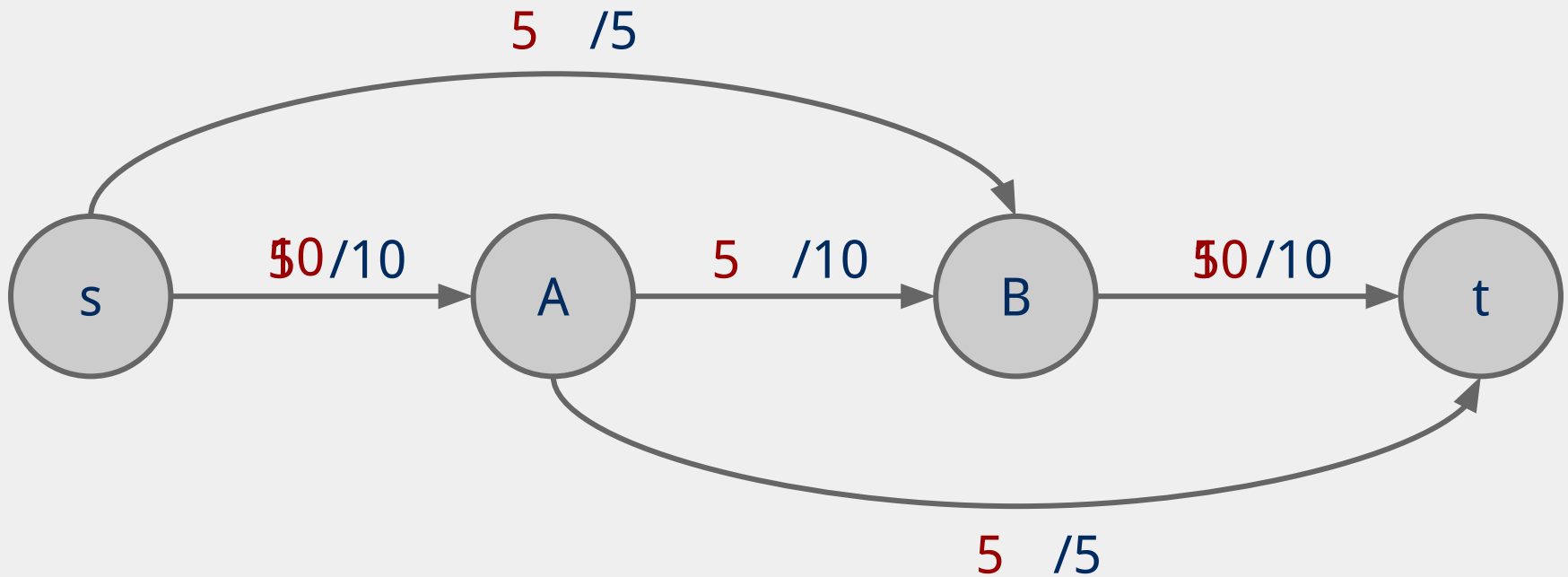
Flow

- Let the $f(u, w)$ be the amount of flow being carried along the edge (u, w)
- Some rules on the flow running through an edge:
 - $\forall (u, w) \in E \ f(u, w) \leq c(u, w)$
 - $\forall u \in (V - \{s, t\}) \ (\sum_{w \in V} f(w, u) - \sum_{w \in V} f(u, w)) = 0$

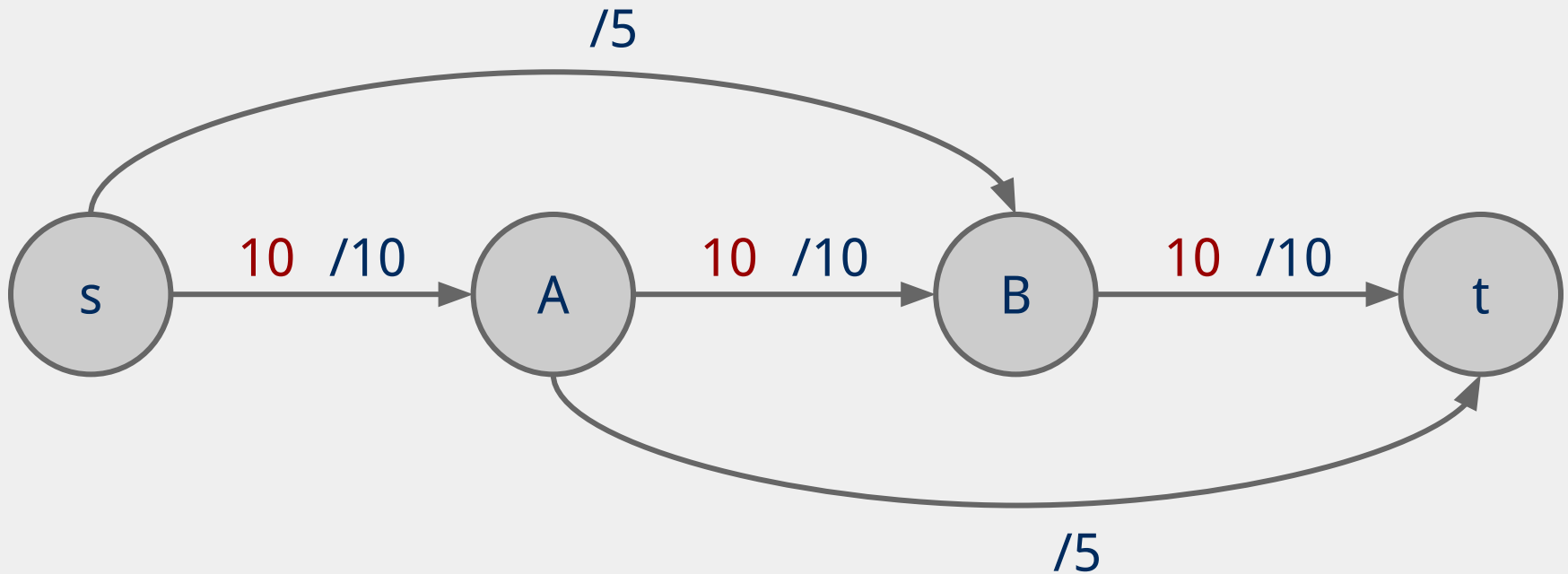
Ford Fulkerson

- Let all edges in G have an allocated flow of 0
- While there is path p from s to t in G s.t. all edges in p have some *residual capacity* (i.e., $\forall (u, w) \in p \ f(u, w) < c(u, w)$):
 - (Such a path is called an *augmenting path*)
 - Compute the residual capacity of each edge in p
 - Residual capacity of edge (u, w) is $c(u, w) - f(u, w)$
 - Find the edge with the minimum residual capacity in p
 - We'll call this residual capacity *new_flow*
 - Increment the flow on all edges in p by *new_flow*

Ford Fulkerson example



Another Ford Fulkerson example



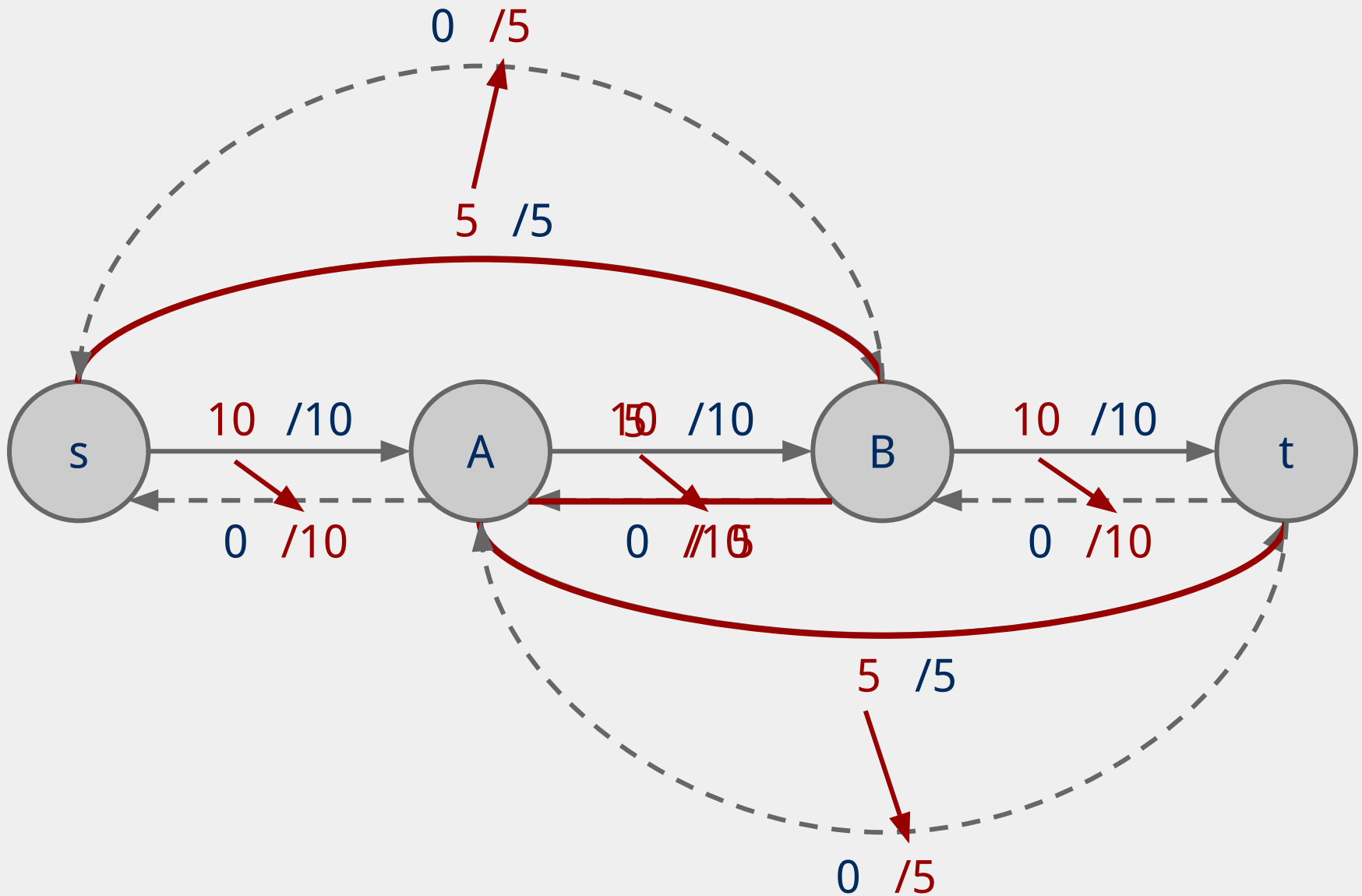
Expanding on residual capacity

- To find the max flow we will have need to consider re-routing flow we had previously allocated
 - This means, when finding an augmenting path, we will need to look not only at the edges of G , but also at *backwards edges* that allow such re-routing
 - For each edge $(u, w) \in E$, a backwards edge (w, u) must be considered during pathfinding if $f(u, w) > 0$
 - The capacity of a backwards edge (w, u) is equal to $f(u, w)$

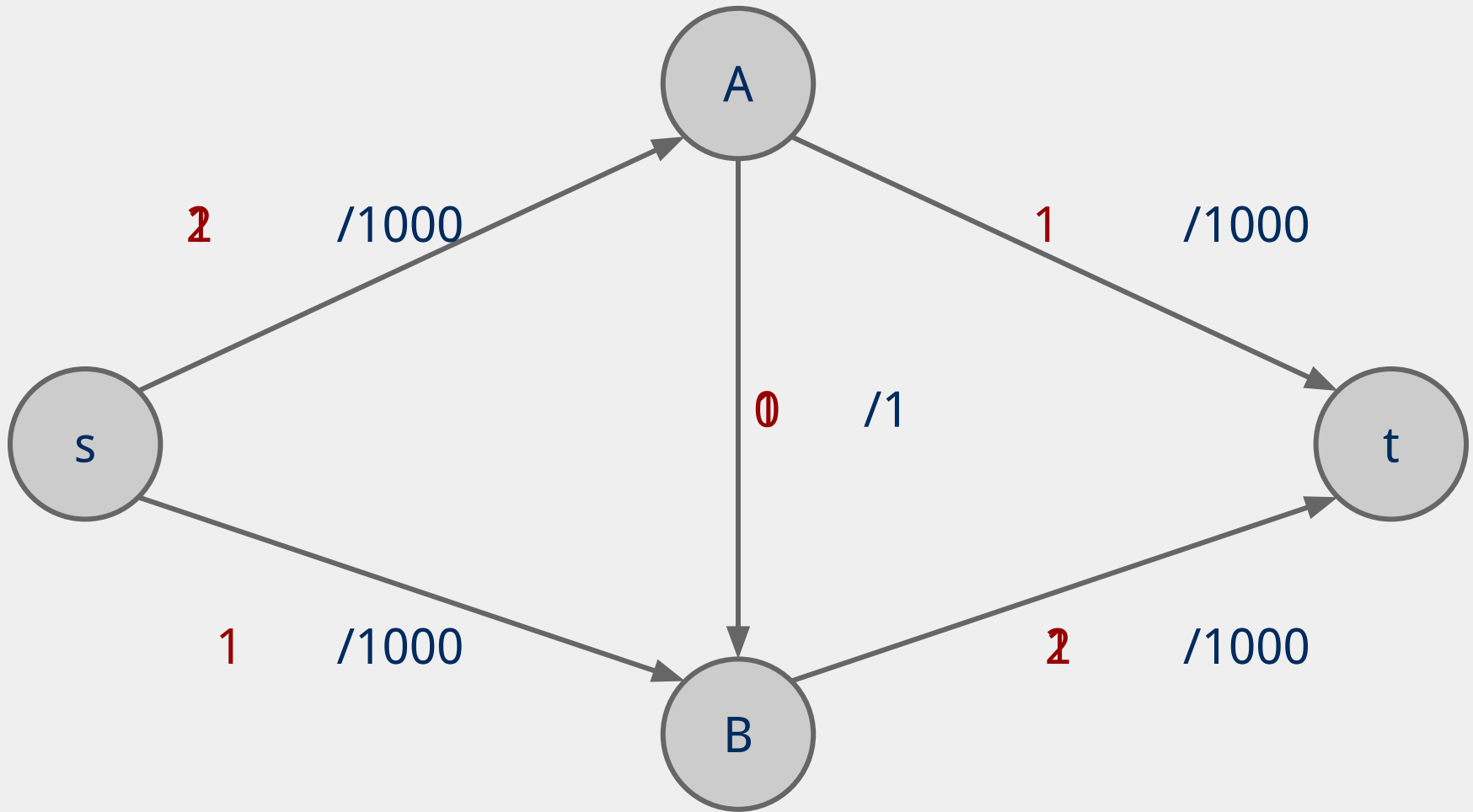
The residual graph

- We will perform searches for an augmenting path not on G , but on a residual graph built using the current state of flow allocation on G
- The residual graph is made up of:
 - V
 - An edge for each $(u, w) \in E$ where $f(u, w) < c(u, w)$
 - (u, w) 's mirror in the residual graph will have 0 flow and a capacity of $c(u, w) - f(u, w)$
 - A backwards edge for each $(u, w) \in E$ where $f(u, w) > 0$
 - (u, w) 's backwards edge has a capacity of $f(u, w)$
 - All backwards edges have 0 flow

Residual graph example



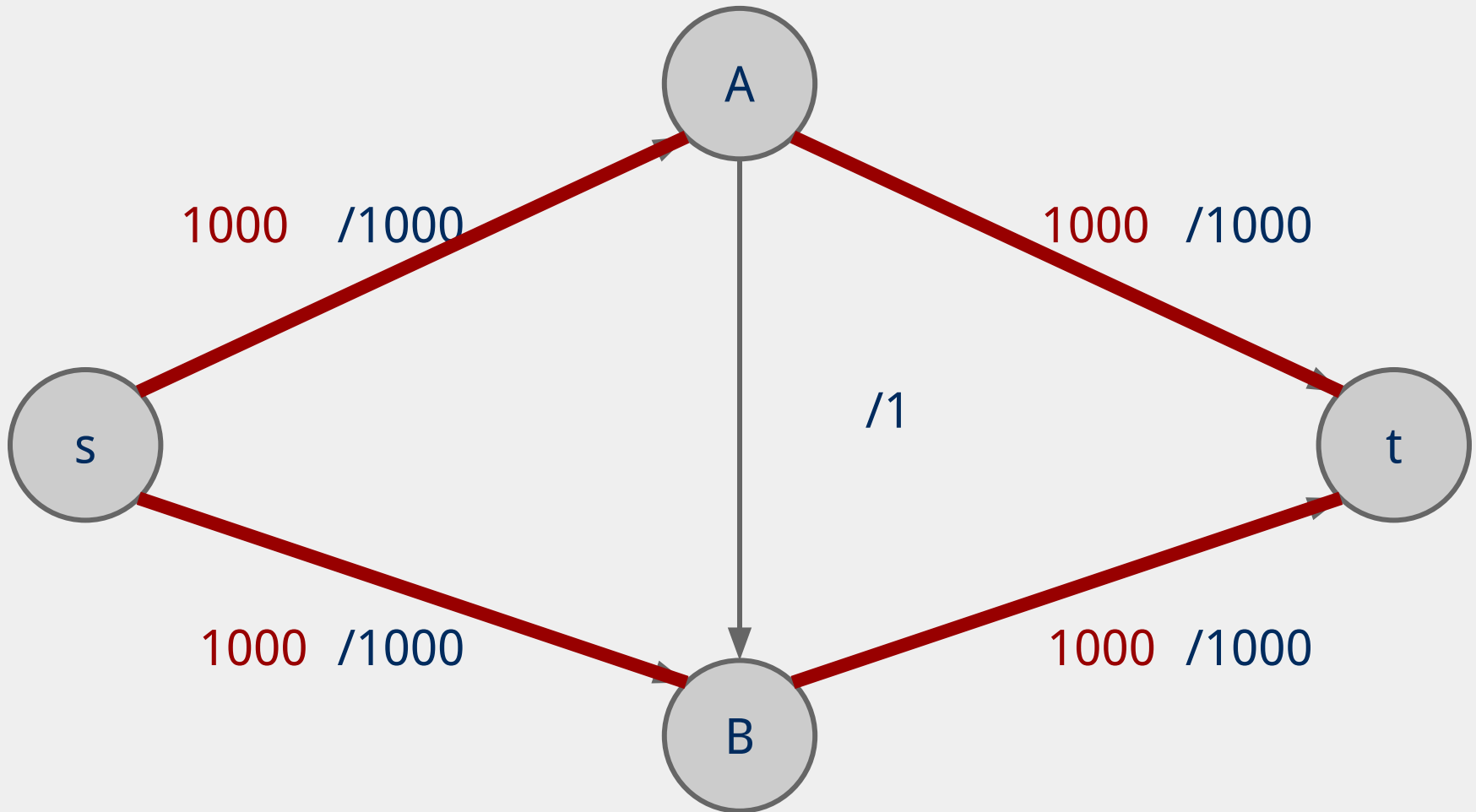
Another example



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?

Implementation concerns

- Representing the graph:
 - Similar to a directed graph
 - Can store an adjacency list of directed edges
 - Actually, more than simply directed edges
 - Flow edges

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)

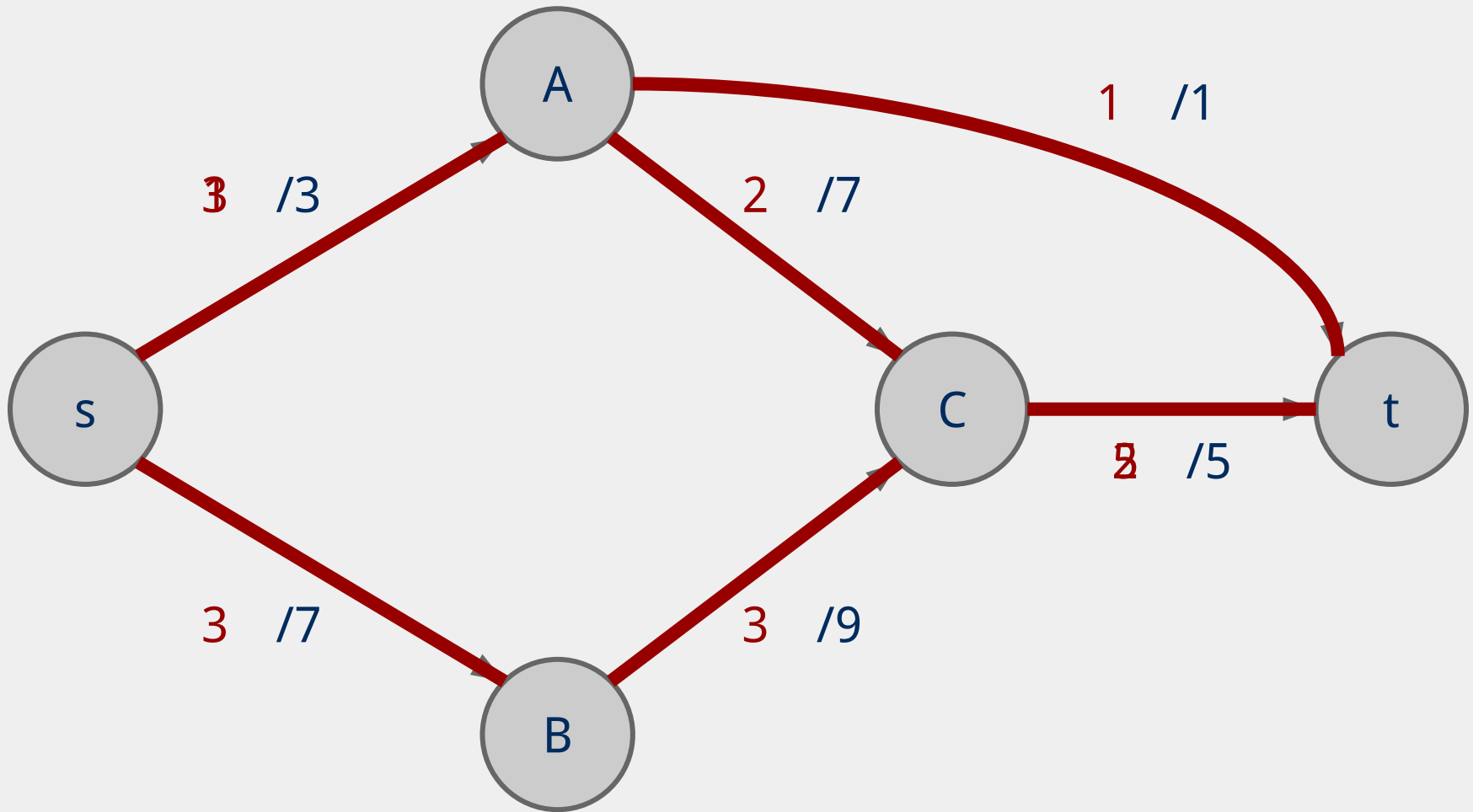
```
edgeTo = [|V|]
marked = [|V|]
Queue q
q.enqueue(s)
marked[s] = true
while !q.isEmpty():
    v = q.dequeue()
    for each (v, w) in AdjList[v]:
        if residualCapacity(v, w) > 0:
            if !marked[w]:
                edgeTo[w] = e;
                marked[w] = true;
                q.enqueue(w);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

Each FlowEdge object is stored
in the adjacency list twice:

Once for its forward edge
Once for its backwards edge



An example to review



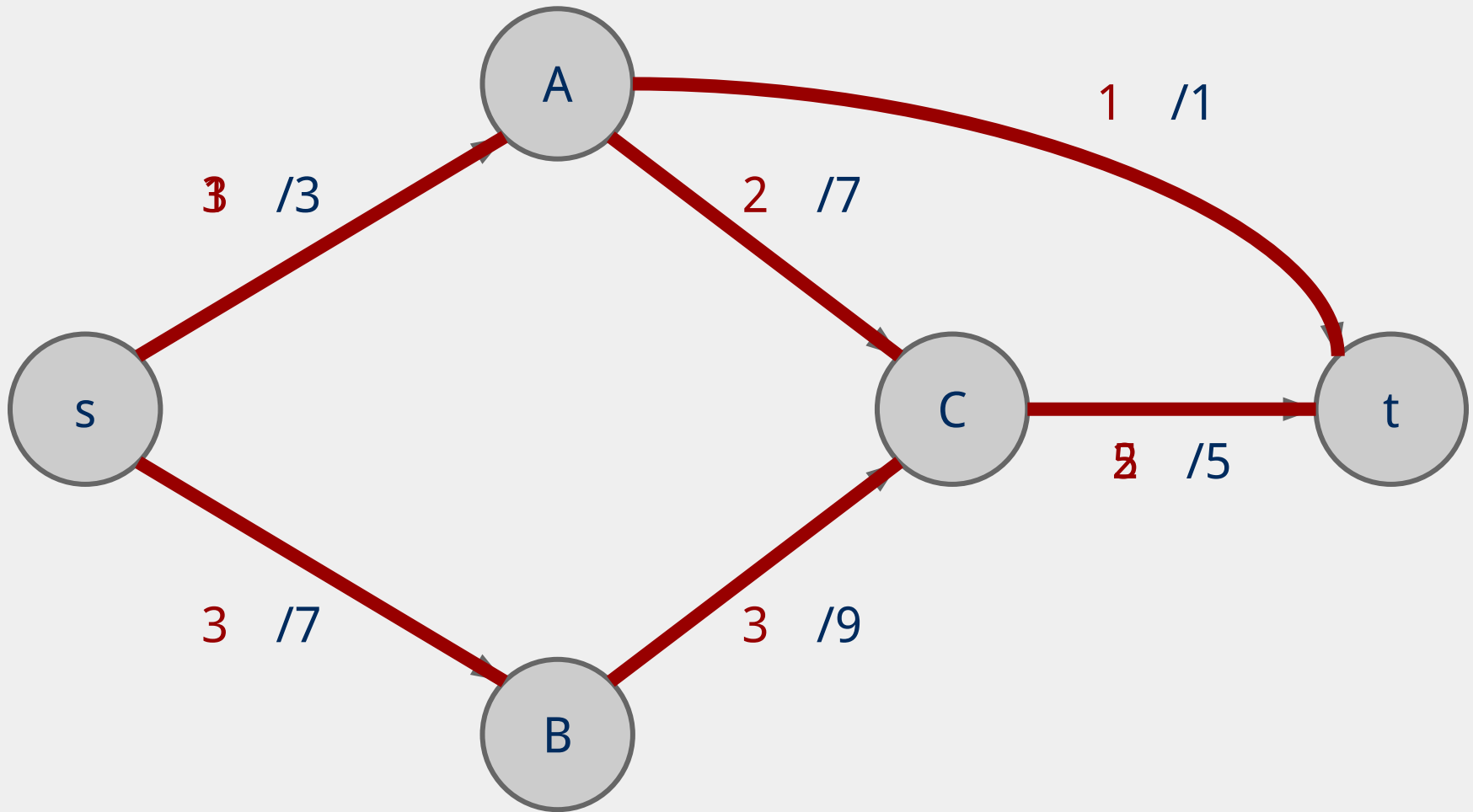
Let's separate the graph

- 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



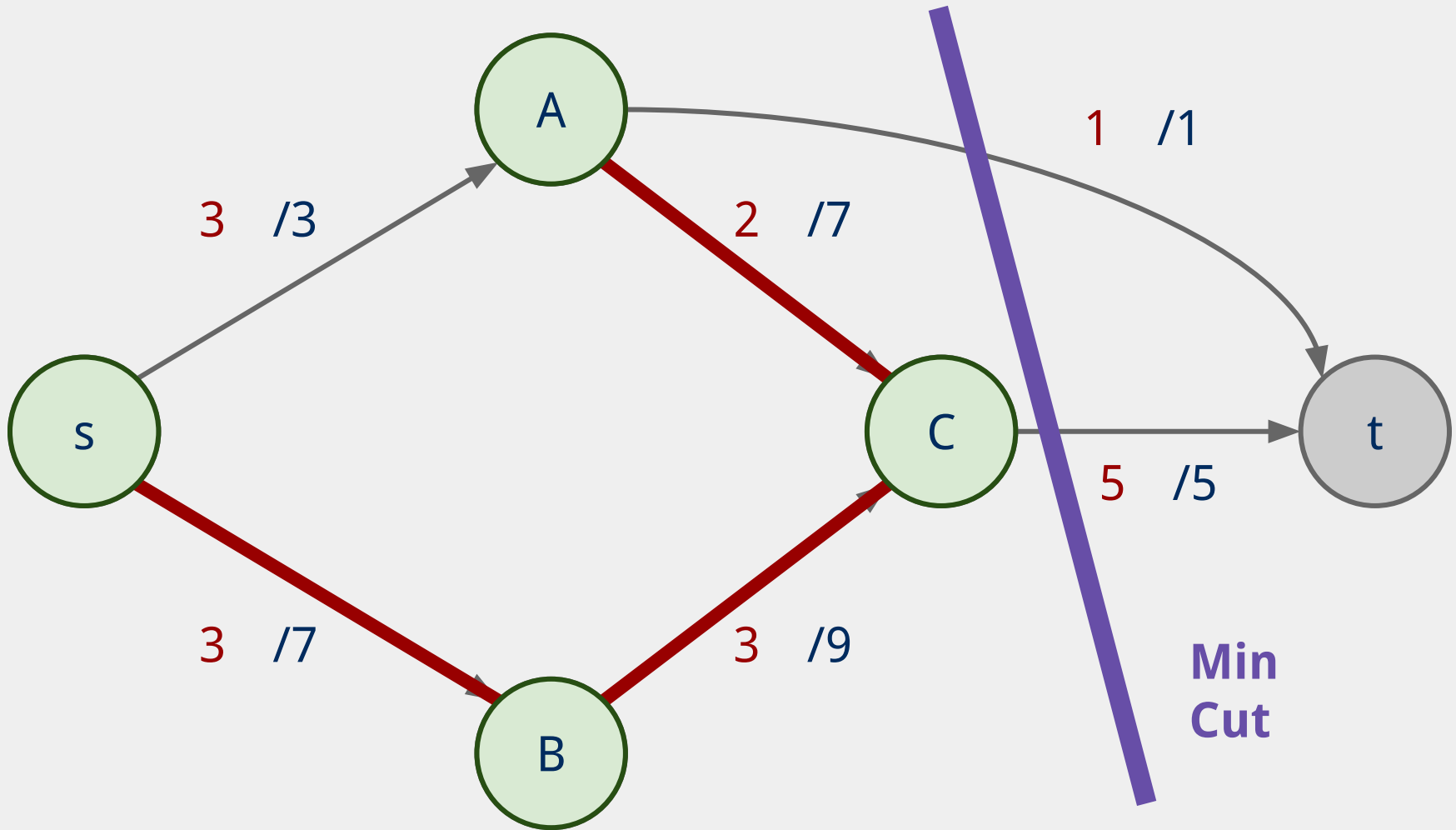
Max flow == min cut

- 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

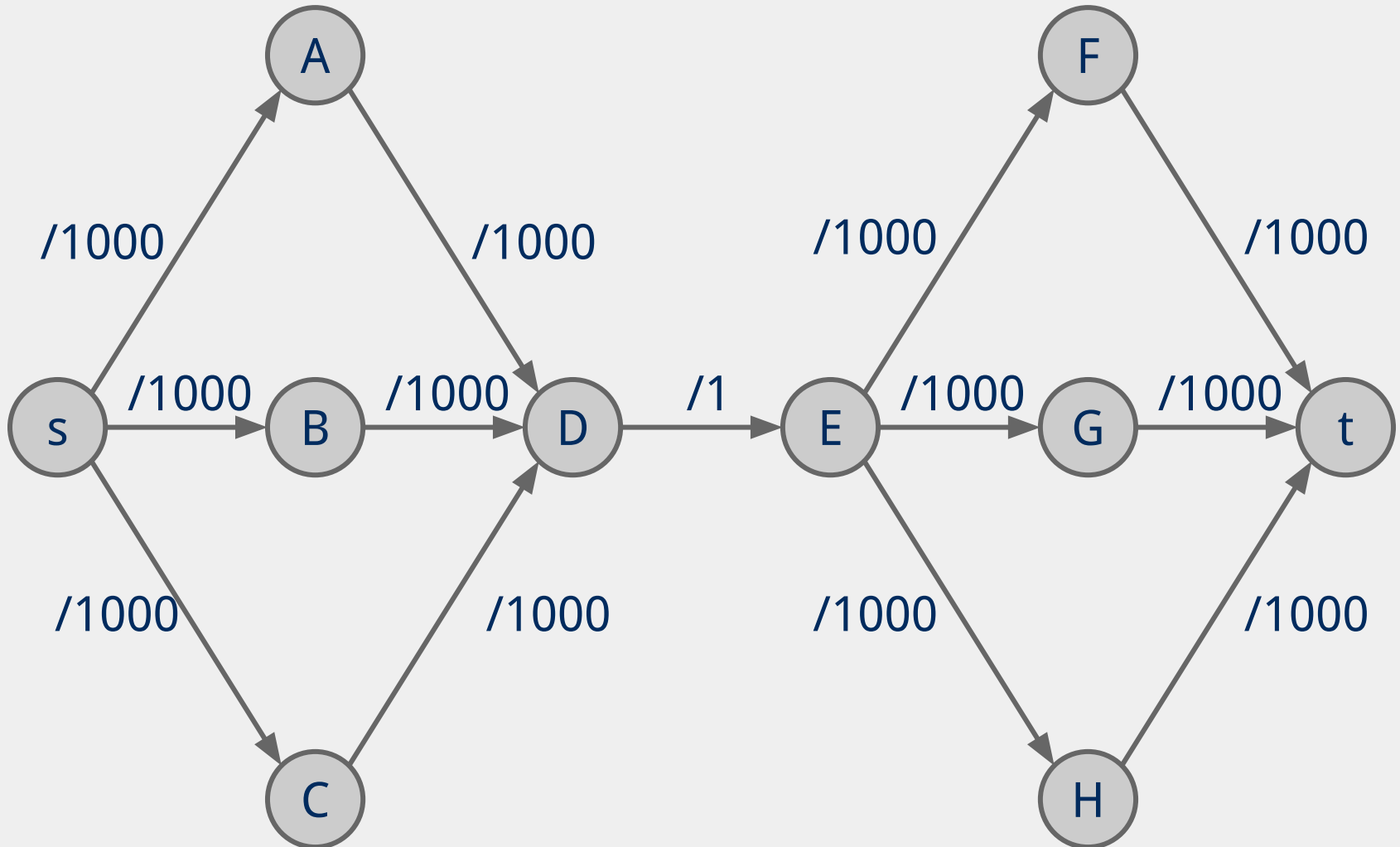
Determining a minimum st-cut

- 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



Will max flow/min cut always be near s/t?



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

