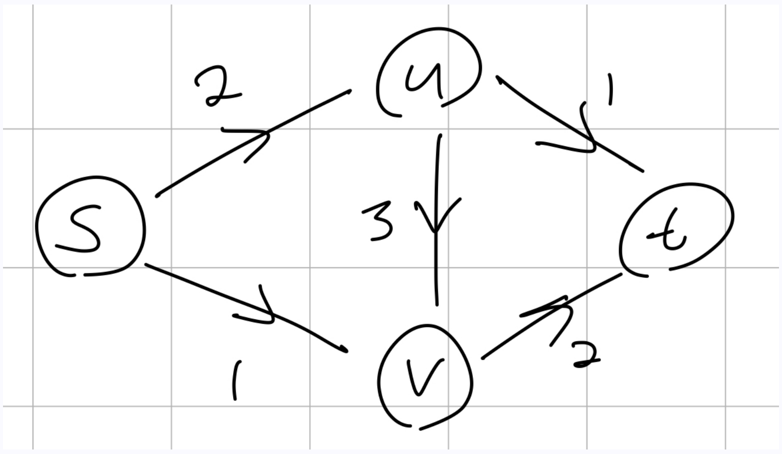
19 Network Flow

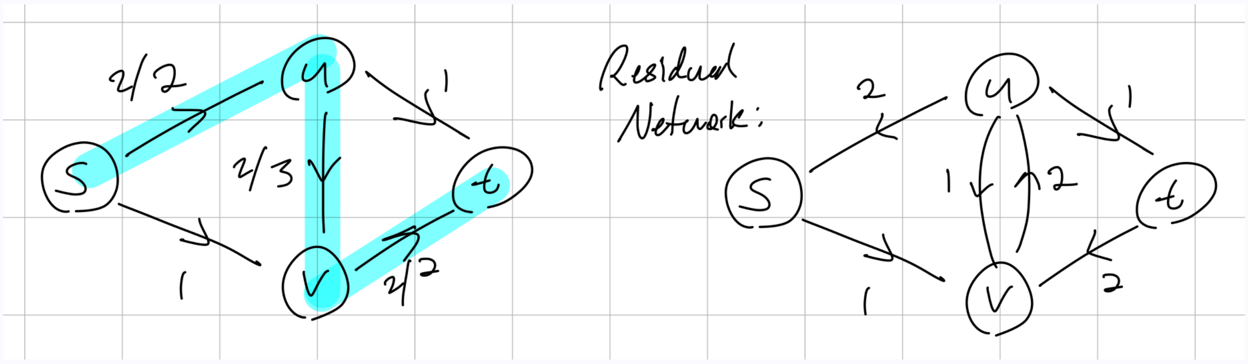
19.1 Flow Networks

Example 19.1

Take the following flow network from last lecture:



Given the example flow from last class, the residual flow network will be the following:



Definition 19.2

An **augmenting path** is a simple path from s to t in the residual network with positive capacity.

19.1.1 Ford-Fulkerson Algorithm

How do we find an augmenting path? We can just use BFS.

How do we augment? For a path P and flow f, let bottleneck(P, f) be the smallest capacity of any edge on P in the residual network corresponding to f.

```
Augment(f, P):
    let b = bottleneck(P, f)
    for each edge e = (u, v) along P
        if e is a forward edge
            increase f(e) by b
        else (e is a backward edge)
            decrease f((v, u)) by b
        endif
```

Why does Augment(f, P) produce a valid flow?

- If e is a forward edge, we increase the flow by b. Every edge along P has residual capacity at least b, so we satisfy the capacity constraint.

- If e is a backward edge, then we decrease the flow by b (on the original network), which is at most $f(e)$, so the resulting flow is non-negative.
- Whenever we add flow into an internal (not source or sink) vertex, we add the same flow going out.

```
MaxFlow(G, c, s, t):
    Let f(e) = 0 for all edges e of G
    While there is a simple path P from s to t
      in the residual network of flow f
        Update f to Augment(f, P)
        update residual network to use new flow f
    Endwhile
    return f
```

19.1.2 Termination and Running Time

Lemma 19.3
The value of the flow strictly increases at every step of the algorithm.

Proof. The first edge of the augmenting path P starts from s. The flow along this edge is increased by $b = \text{bottleneck}(P, f) > 0$. Since P is simple, this is the only edge of P that involves s, so the value of the flow is increased by b. □

Lemma 19.4
At each step of the algorithm, the flow values and residual capacities are all integers.

Proof. This is true initially. Every step just involves adding/subtracting flows and capacities, so this remains true for the whole algorithm. □

Thus, the value of the flow increases by at least 1 at every step.

The largest possible flow value is at most

$$C = \sum_{e \in E \text{ leaving } s} c_e$$

So, the algorithm goes through the loop at most C times.

Since each pass through the loop can be implemented in time $O(m+n)$, the overall running time is $O(C \cdot (m+n))$.

19.1.3 Maximum Flows and Minimum Cuts

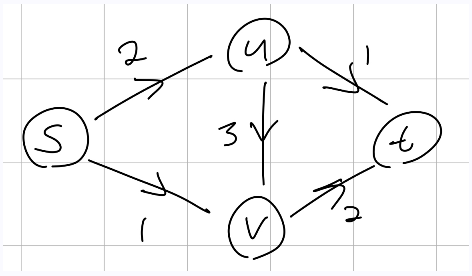
A **cut** in $G = (V, E)$ is a bipartition of V ($V = A \cup B$, $A \cap B = \varnothing$).

This is an $s - t$ cut if $s \in A$ and $t \in B$.

In a flow network, the **capacity** of an $s - t$ cut is the total capacity of all edges leaving A.

$$c(A, B) = \sum_{e \text{ out of } A} c_e$$

Example 19.5
Take the flow network from before:



The cut $A = \{s\}$ has capacity 3.
The cut $A = \{s, u, v\}$ has capacity 3.
The cut $A = \{s, u\}$ has capacity 5.
The cut $A = \{s, v\}$ has capacity 4.