Max Flow & Applications

Frank

November 7, 2021

1 Max Flow

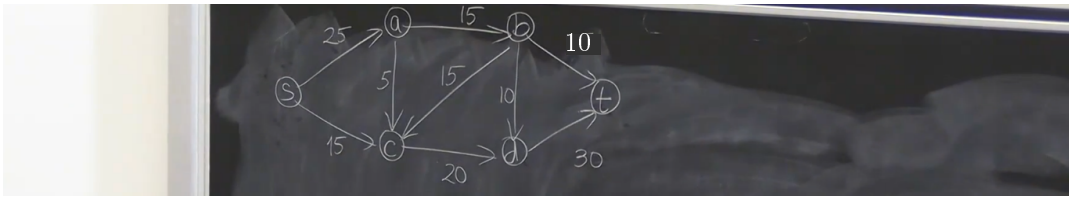
1.1 Introduction

- Define Flow Network $F = (G, s, t, c)$
  - Where $G = (V, E)$ is a directed graph
  - s = source, t = destination $\in V$

    No edges into s and no edges out of t
  - $c : E \to \mathbb{R}^{\geq 0}$

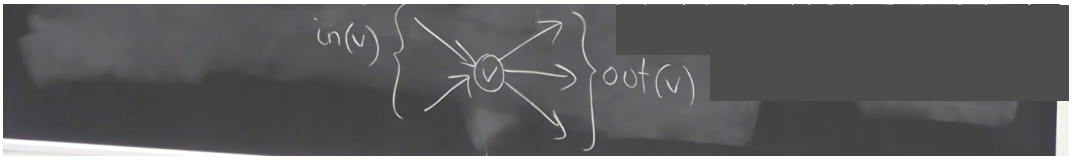
    $c(u, v)$ = capacity of $(u, v) \in E$

  Flow network example:

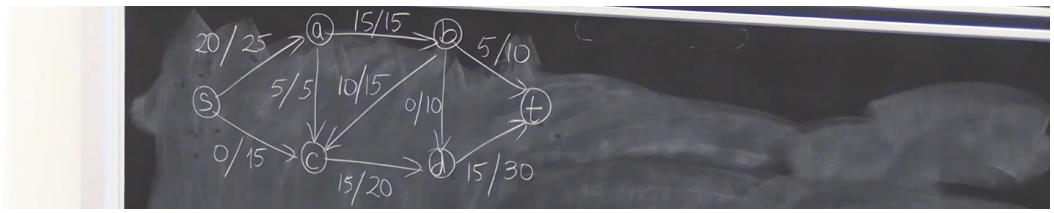


- Define a Flow f in F
  - $f : E \to \mathbb{R}^{\geq 0}$ s.t.

  1. [capacity] $\forall e \ 0 \leq f(e) \leq c(e)$
  2. [conservation] $\forall v \in V - \{s, t\}$, $\sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e)$



Flow example:



Defined as flow/capacity

- Define Value of Flow f, $V(f) = \sum_{e \in out(s)} f(e)$.

  In the above example, we see that a total flow of 20 goes out from s. So $V(f) = 20$

1.2 Max Flow Problem

- **Input:** $F = (G, s, t, c)$
- **Output:** Flow f in F of maximum value: $\forall$ flow f' in F, $V(f) \geq V(f')$

1.3 Minimum Cut Problem

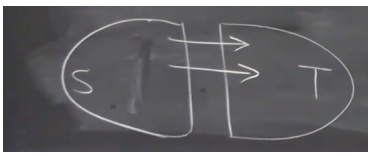
1.3.1 Definition

- Define Cut of $F = (G, s, t, c)$ is a pair (S, T) s.t. $S, T \subseteq V$, $s \in S, t \in T$, $S \cap T = \emptyset$, $S \cup T = V$

  For example, with the above flow network, a cut would be $S = \{s, c\}$, $T = \{a, b, d, t\}$

- Capacity of cut (S, T): $c(S, T) = \sum_{e \in out(S) \cap in(T)} c(e)$, where $out(S) = \cup_{v \in S} out(v)$ and $in(S) = \cup_{v \in S} in(v)$

  In layman terms, this means the capacity of all the edges that are going out from S into T, which we defined as the cut.

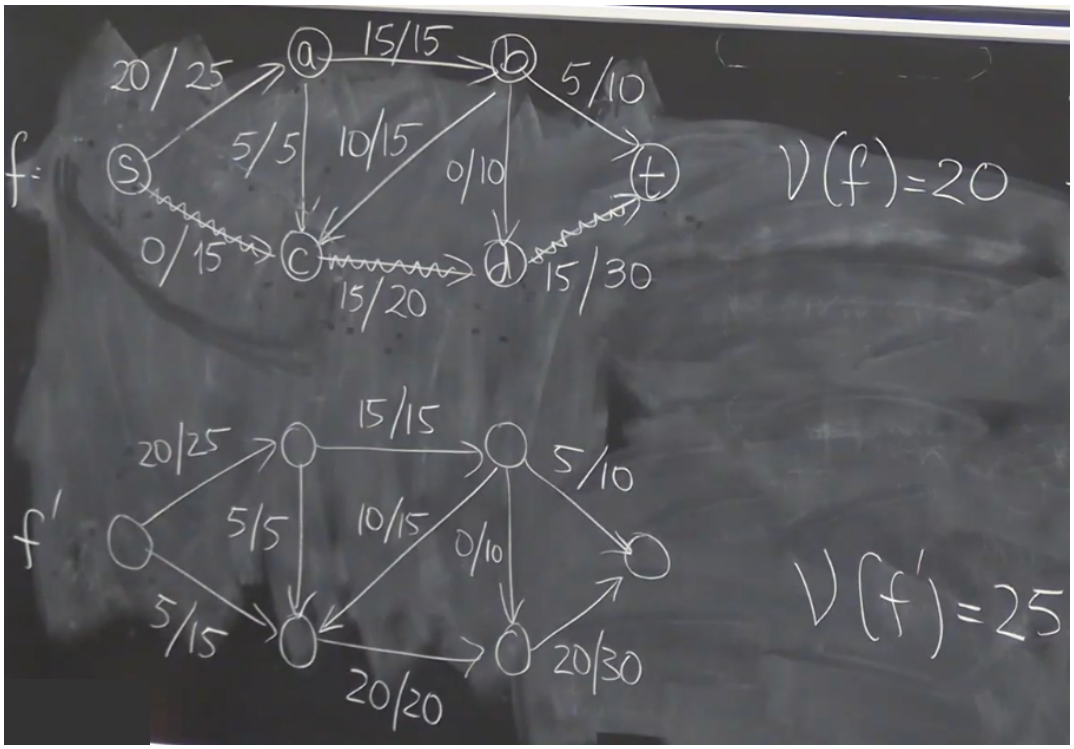


1.3.2 Problem

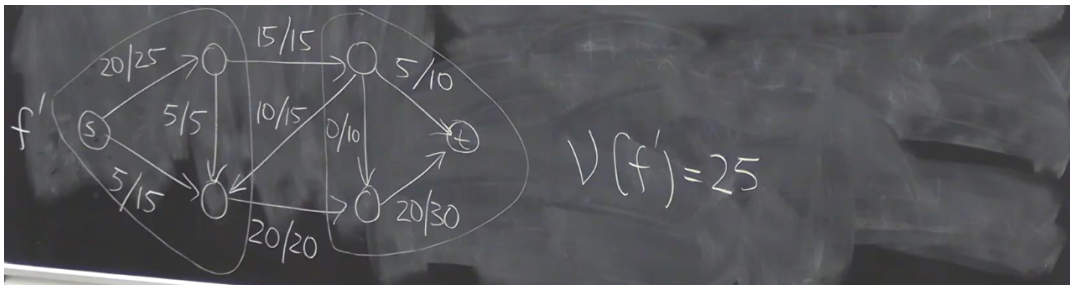
- **Input:** $F = (G, s, t, c)$
- **Output:** Cut (S, T) of minimum capacity: $\forall (S', T')$, $c(S, T) \leq c(S', T')$ cut

1.4 Back to the Max Flow Problem

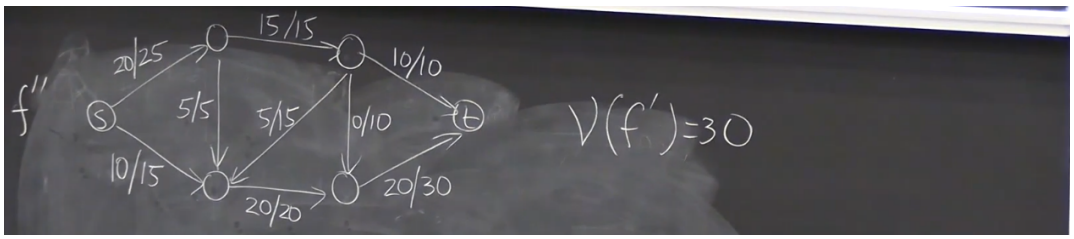
- Was our original flow the max flow? No, consider:



We pushed more into the path. But however, f' is still not optimal, because if we consider the cut, there is backwards flow:

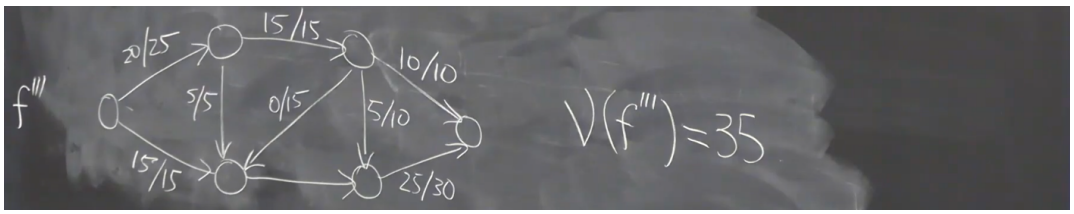


- By redirecting the flow from $c \to b$, we can construct the following flow:

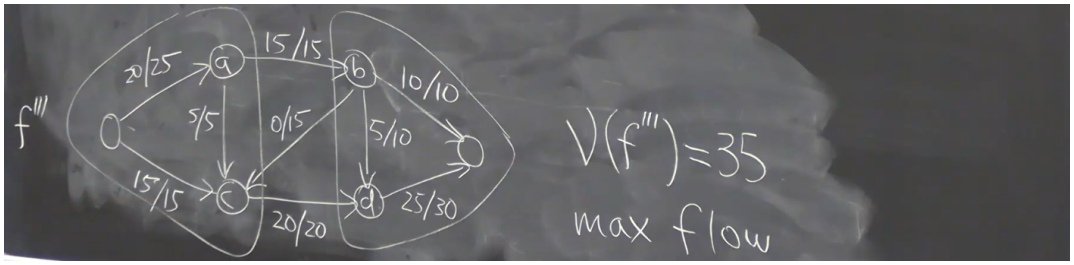


  Note that we can redirect flows as long as we satisfy the conservation of flow.

- Note that we could redirect again from $c \to b$ to end up with an optimal flow:



  To see that this is indeed optimal, look at the following cut,

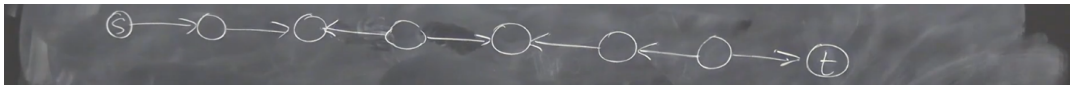


  We have that the cut's capacity is filled up, thus it is impossible to send anymore flow through this cut. This is because in order for flow to travel from s to t, it has to pass through the cut, and we have a cut capacity of 35, so the flow is limited at 35.

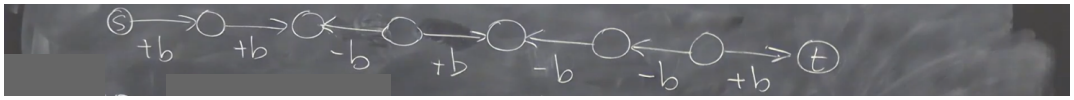
1.5 Solving the Max Flow Problem

Consider the following steps [**Path augmentation**]:

- Find simple (no node is repeated) $s \to t$ "path" [ignoring direction]



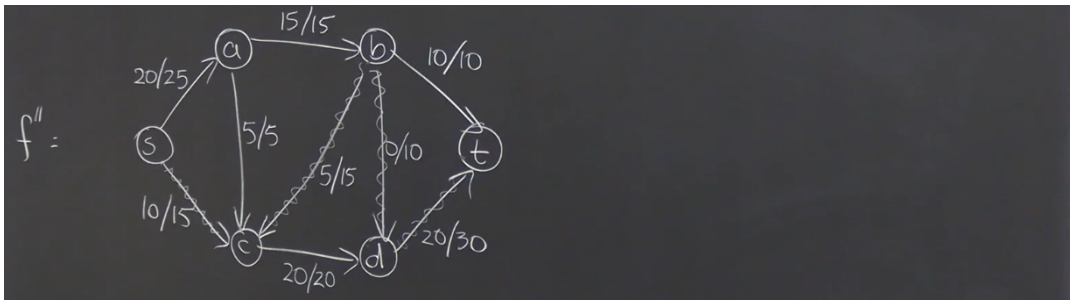
- Increase traffic on each edge traversed forward by some amount b.
- Decrease traffic on each edge traversed backward by the same amount b.



Following these steps, we can increase our flow by pushing an additional **bottle-neck amount** b thourgh the flow, by taking adding to positive edges and taking away from negative edges.

Note that we will choose the smallest b on the path, because otherwise we will overflow or underflow the capacities.

Note that the "redirection" we performed earlier is simply path augmentation of a "path":



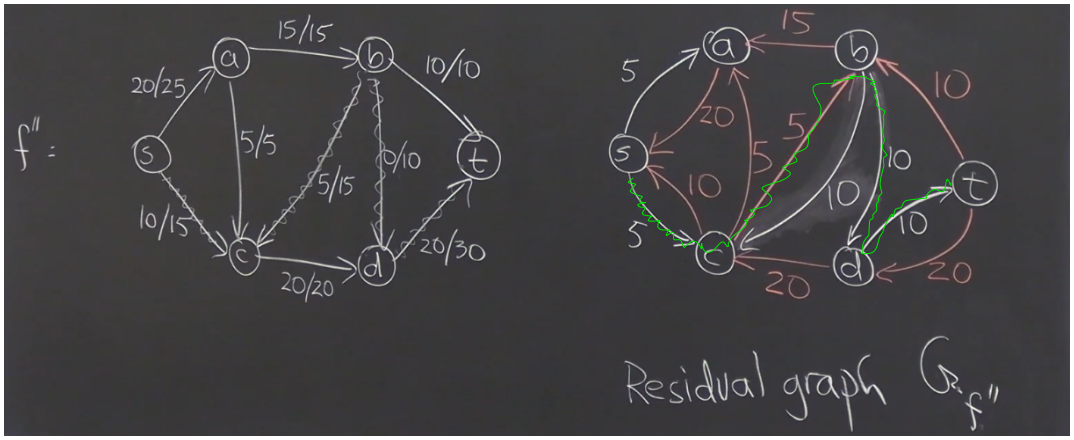
Note in this case we simply found a "path" and pushed forward "forward edges" and pushed back "backward edges" on the "path"

1.6 Residual Graph $G_{f''}$

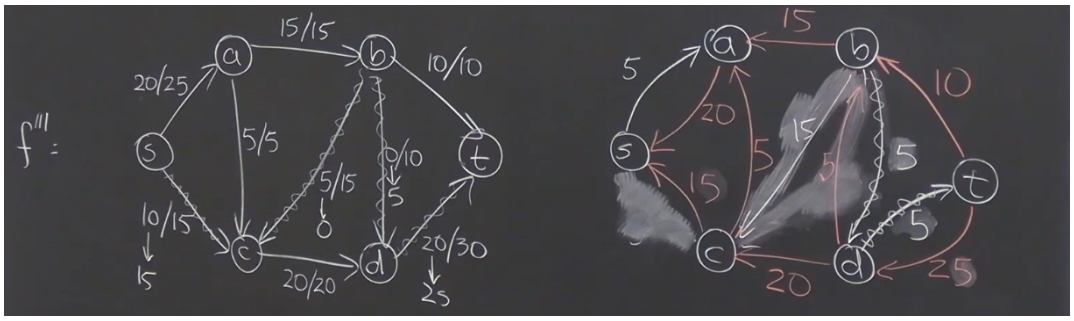
1.6.1 Introduction

For every edge in G, create 2 additional paths (if possible), one forward (remaining capacity) and one back (capacity able to be pushed back)

For example, for our original graph, we will have residual graph $G_{f''}$:



Then now we can actually follow a path instead of a "path" to push through from $s \to t$, of course the path being a combination of both type of edges from $G_{f''}$



When we have no more paths from s to t, then we cannot use this method to improve our flow, and will prove that our flow is **optimal**.

Note again that we will use the minimum residual capacity edge on the path to push through to avoid overflow or underflow of the edges, to improve the flow by b

1.6.2 Algorithm for path augmentation

- Given $F = (G, s, t, c)$ and flow f in F:

  Residual graph G_f of f wrt F: $G_f = (V, E_f)$ with edges labeled by residual capacities $c_f : E_f \to \mathbb{R}^{>0}$

- For each edge $e = (u, v)$ in G s.t. $f(e) < c(e)$ (Unsaturated), add to E_f, $e = (u, v)$ with $c_f(u, v) = c(e) - f(e)$ (**Forward edge**)

- For each edge $e = (u, v)$ in G s.t. $f(e) > 0$, add to E_f the edge (v, u) with $c_f(v, u) = f(e)$ (**Backward edge**)

1.7 Algorithm

Given $F = (G, s, t, c)$, flow f, G_f

Let p = simple $s \to t$ path in G_f

1: **procedure** AUGMENT(f, p)
2: $b \leftarrow$ min residual capacity of edges on p in G_f
3: **for** (each edge on p) **do**
4: **if** ((u, v) is FW) **then**
5: $f(u, v) \leftarrow f(u, v) + b$
6: **else**
7: $f(v, u) \leftarrow f(v, u) - b$
8: **return** f ▷ improved flow

- **Lemma 1:** If f is a flow in F then $f' = \text{augment}(f, p)$ is
  1. a flow in F and
  2. $V(f') > V(f)$

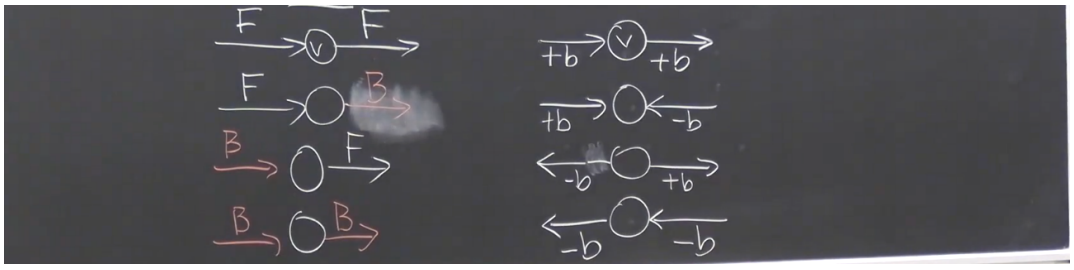
  Proof: (1) prove that f' satisfies capacity(a) + conservation(b)

  a) capacity:

  consider edges on path p (relevant) in G_f
  If the edge is forward, then we will add b, and if the edge is backward, then we will subtract b, in either case since b is the smallest in absolute value on p, we will satisfy capacity.

  b) conservation:

  p is simple $\implies \forall v \neq s, t$ on p, we have one edge into v and one edge out of v. Then we have 4 cases:



  For which we have retained the conservation of flow in all cases, either by increasing/decreasing the flow by the same amount or by canceling out.

  Proof: (2) $V(f') = V(f) + b^{>0} > V(f)$, note $V(f)$ is the total value of the outgoing edges of s

1.8 Ford-Fulkerson Algorithm for Max Flow

```
1: procedure MaxFlow(F(G, s, t, c))
2:     for (each edge (u, v) of G) do
3:         f(u, v) ← 0                          ▷ initialize 0 because want max flow
4:     construct residual graph G_f
5:     while (G_f has an s → t path) do
6:         p ← some simple s → t path of G_f     ▷ any order of choosing p
7:         f ← augment(f, p)
8:         update G_f
9:     return f
```

1. Does FF terminate?

   Yes, if capacities are **integers**. If we have irrational numbers, we may endlessly improve the flow, and the flow may even converge to the incorrect value.

   So assume that all capacities are integers from now on.

2. If it does, does it return max flow?

   Yes, (to be shown)

1.9 Termination & Running-Time of FF

1.9.1 Termination

- At the end of each iteration of FF, each edge has integer traffic. If we assume capacities are integers
- $C = \sum_{e \in out(s)} c(e) \implies \forall$ flow f, $V(f) \leq C$, then C is an upper bound on the maximum flow.

  $\implies$ FF terminates after at most C iterations

1.9.2 Running-Time

- Assume $m = \Omega(n)$, m dominates n

  Then line 5-6 takes $O(m)$

  Line 7 takes $O(m)$

  Line 8 takes $O(n)$

- So each iteration takes $O(m)$, and there is at most C iterations

  $\implies$ FF runs in $O(mC)$ time (pseudo-polynomial)

Particularly, if we are not careful in choosing our augmented-path, we can actually achieve $O(mC)$ quite easily

Consider this example where we repeatedly choose p such that the edge (a, b) is always contained and where we alternate between $s \to a \to b \to t$ and $s \to b \to a \to t$, we will actually take C iterations to achieve the max flow.

1.10 Choice of augmenting paths

1. Widest path: pick the path with maximum b each time.

   Improve number of iterations to $O(m \log C)$ and running-time to $O(m^2 \log C)$ (polynomial)

   Actually picking the widest path left as an exercise (with an algorithm already seen in class).

   Should do this because might be on exam!

2. Choose the shortest path (minimizing number of edges)

   Find with BFS (linear time)

   Improve number of iterations to $O(mn)$ and running-time to $O(m^2 n)$ (Strongly-polynomial)

   Where strongly-polynomial means polynomial not depending on values (if arithmetic ops take O(1) time)

3. Should know both facts, but proofs optional

1.11 Correctness

- **Lemma 2:**

  $\forall$ flow f, $\forall$ cut (S, T)

$$V(f) = \sum_{e \in out(S) \cap in(T)} f(e) - \sum_{e \in out(T) \cap in(S)} f(e)$$

Proof:

$$\begin{aligned} V(f) &= \sum_{v \in S} \left(\sum_{e \in out(v)} f(e) - \sum_{e \in in(v)} f(e) \right) \\ &= \sum_{v \in S} \sum_{e \in out(v)} f(e) - \sum_{v \in S} \sum_{e \in in(v)} f(e) \\ &= \sum_{e \in out(S)} f(e) - \sum_{e \in in(S)} f(e) \\ &= \sum_{e \in out(S) \cap in(T)} f(e) - \sum_{e \in out(T) \cap in(S)} f(e) \end{aligned}$$

- Given Lemma 2, we have an immediate corollary
- **Corollary 3**: $\forall$ flow f, $\forall$ cut (S,T), $V(f) \leq c(S,T)$

  Proof:

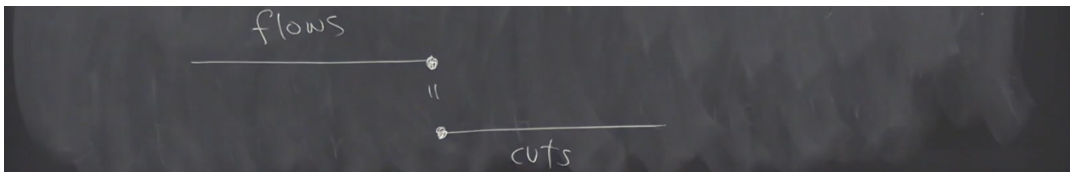
  Note that

$$\begin{aligned} V(f) &= \sum_{e \in out(S) \cap in(T)} f(e)^{\leq c(e)} - \sum_{e \in out(T) \cap in(S)} f(e)^{\geq 0} \\ &\leq \sum_{e \in out(S) \cap in(T)} c(e) \\ &= c(S,T) \end{aligned}$$

  Or that the capacity of any flow is less than or equal to the capacity of any cut

  From Cor 3, Cor 4 follows immediately, because if every flow is less than or equal to every cut, then if a flow is equal to a cut, then the cut must be minimal, and the flow must be maximal

- **Cor 4:** $\forall$ flow f, $\forall$ cut (S,T), if $V(f) = c(S,T)$, then
  1. f is max flow
  2. (S,T) is min cut



Follows from Cor 3

- **Proof of FF:**

  f is flow at end of FF (by termination)

  Consider G_f, there is no path from $s \to t$ (by termination condition)

  Let S = set of nodes reachable from s in G_f (includes s), and T = rest of nodes (includes t)

  $\implies (S, T)$ is a cut

  By L2,

$$V(f) = \sum_{e \in out(S) \cap in(T)} f(e) - \sum_{e \in out(T) \cap in(S)} f(e)$$

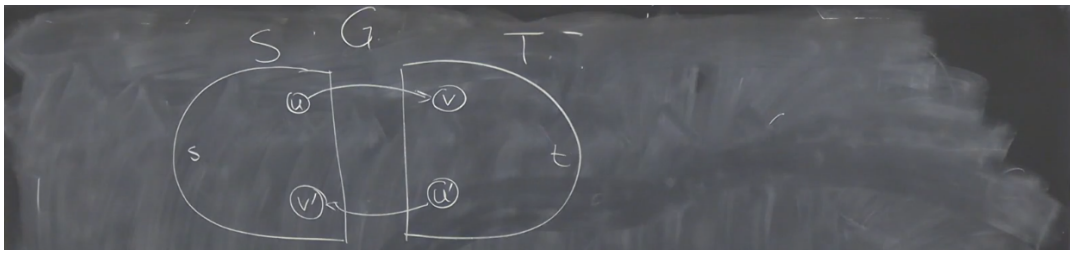
  will prove:

  1. $e = (u, v) \in out(S) \cap in(T) \implies f(u, v) = c(u, v)$ (Saturated)
  2. $e' = (u', v') \in out(T) \cap in(S) \implies f(u, v) = 0$

     $\implies V(f) = c(S, T)$

     $\implies f$ is max, (S, T) is min (Cor 4)

  Consider the actual network G



  1. $f(u, v) = c(u, v)$, because if $f(u, v) < c(u, v)$, G_f contains FW edge (u, v) then $v \in S \implies$ contradiction to the fact that $v \in T$
  2. $f(u', v') = 0$, because if $f(u', v') > 0$, G_f contains BW edge $(v', u') \implies u' \in S$, which contradicts $u' \in T$

  Thus FF returns the max flow and min cut

1.12 Min Cut Algorithm

1. $f \leftarrow$ MaxFlow(F)
2. $S \leftarrow$ set of nodes reachable from s in G_f

   $T \leftarrow$ rest of the nodes
3. return (S, T)

2 Applicable Theorems

2.1 Max-Flow Min-Cut Theorem

In any flow network F, value of max-flow in F is equal to the capacity of minimum cut in F.

Proven by us

2.2 Integrality Theorem

If capacities are integers, then there exists a maximum flow s.t.
every edge has an integer traffic, called **integral flow**