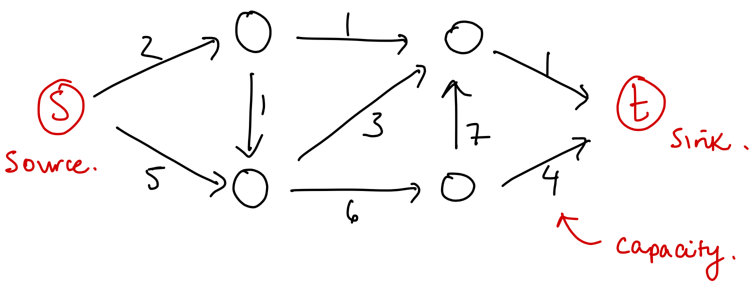
Flow Networks

A flow network is a directed graph $G = (V, E)$ and:

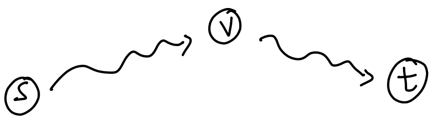
- a specified source vertex s
- a sink vertex t
- and a given capacity, $c(u,v) \geq 0$, for each edge $(u,v) \in E$.

<u>EX</u>



We assume:

- each vertex lies on a path from s to t
- no self-loops!

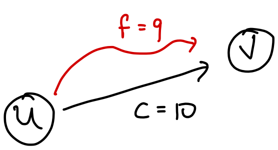


for edges that don't exist,

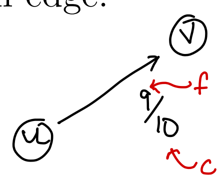
- if edge $(u,v) \notin E$, then set $c(u,v) = 0$.

Flow function:

a function that defines the *flow* across each edge.



written as



The flow function can be defined as *non-negative*, (as in the textbook).

$\boxed{f(u,v) \geq 0.}$ ex. $u \xrightarrow{f=5} v$ Positive net flow from u to v.

The opposite flow from v to u is denoted $f(v,u)$.

<u>ex</u> $u \xleftarrow{f=3} v$ if positive flow goes $v \to u$.

The flow function can also be defined as the *net-flow* between 2 vertices, in which case

$$f(u,v) = -f(v,u)$$ } negative flow goes the other way.

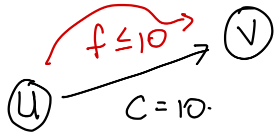
In this definition, we assume if $u \xrightarrow{5} v$

then this implies $u \xleftarrow{-5} v$.

In the following notes, we use the notation which assumes that $f(u,v) \geq 0$.

In either definition, the flow function must be **bounded** and **conservative** at each node:

- bounded:

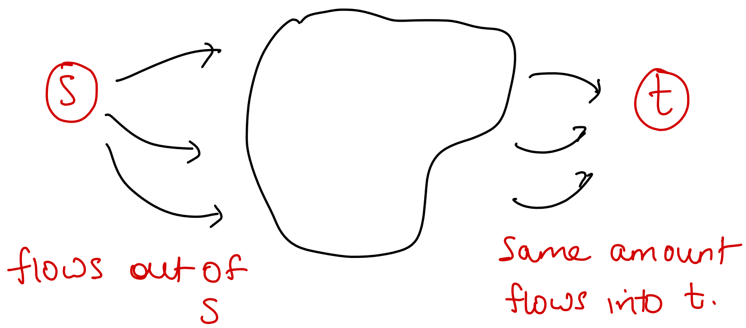


$f(u,v) \leq c(u,v)$

- conservation: in flow = out flow for all vertices except s, t.

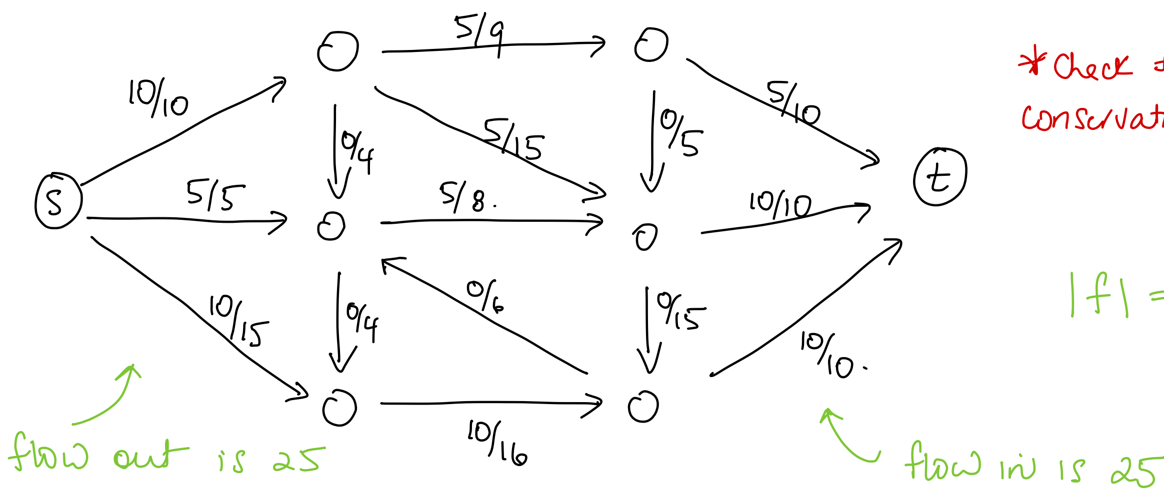
$$\forall u, \ (\notin s,t) \quad \underbrace{\sum_{v \in V} f(v,u)}_{\text{in}} = \underbrace{\sum_{v \in V} f(u,v)}_{\text{out.}}$$

The value of the flow on G**:** How much flow is leaving s and arriving at t:



$$\text{Val}(f) = |f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t).$$

Example:

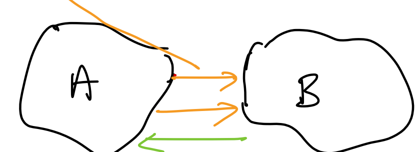


*Check the flow is conservative at each node.

$|f| = 25.$

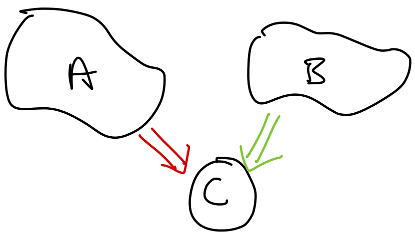
The flow function will be extended to sets A and B in the following way:

- $f(A,B) = \underbrace{\sum_{u \in A} \sum_{v \in B} f(u,v) - \sum_{u \in B} \sum_{v \in A} f(u,v)}_{\text{net flow } A \to B}$



- $f(A,A) = 0$
- $f(A,B) = -f(B,A)$

- $f(s, V) = f(s, V - \{s\})$
- if $A \cap B = \emptyset$, then :



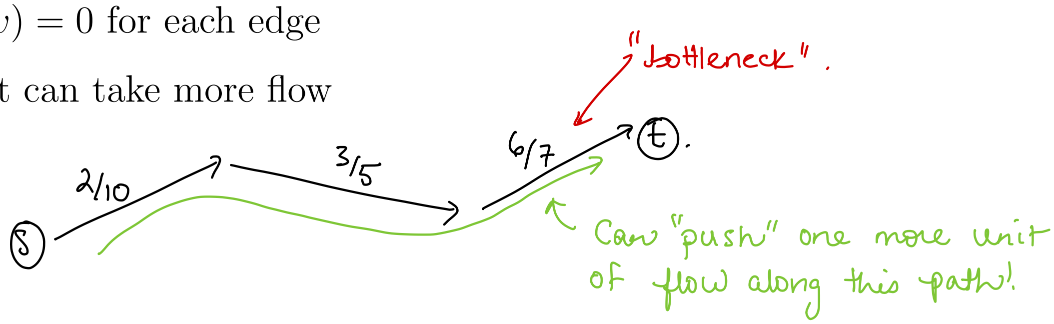
$$f(A \cup B, C) = f(A, C) + f(B, C)$$

Towards a greedy algorithm for finding the maximum flow...

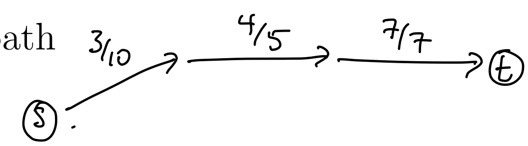
First attempt: start by finding paths along which we can *push* more flow...

- Start with $f(u, v) = 0$ for each edge
- Find a path that can take more flow

ex.

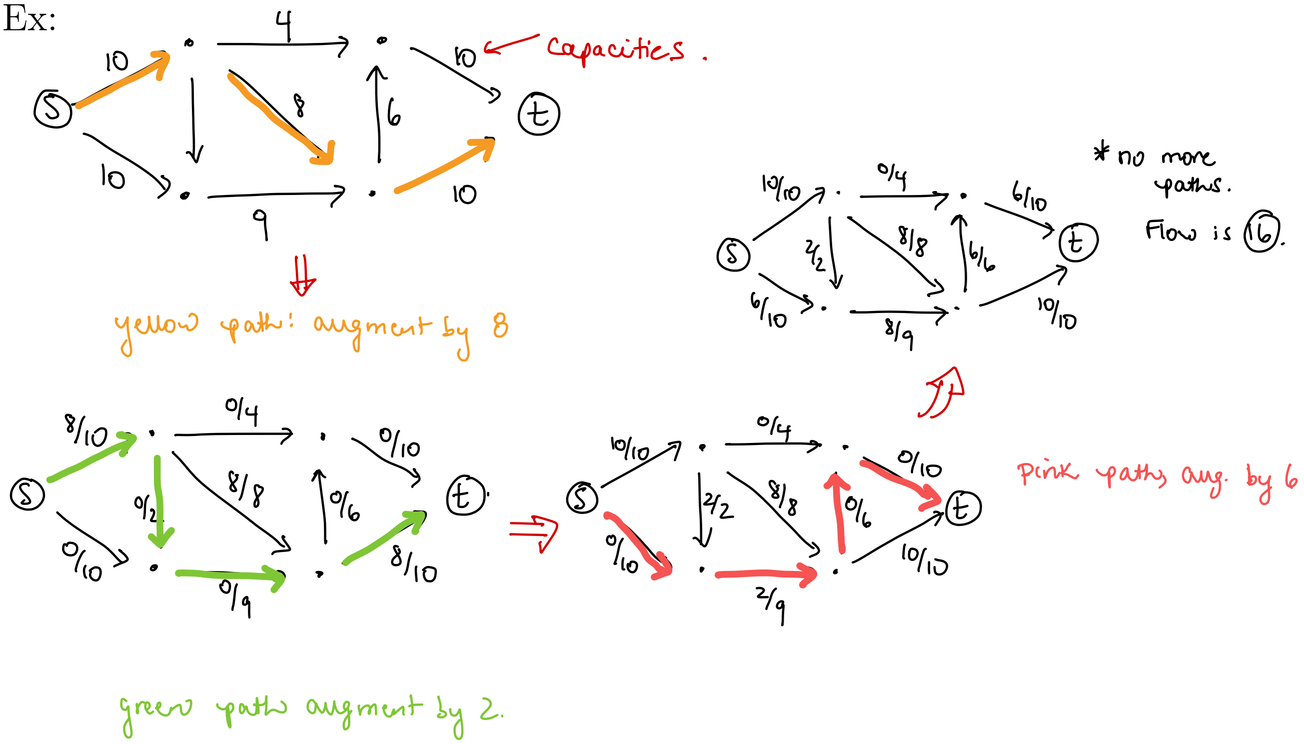


- Augment the flow along that path



- Continue until you get stuck..

Ex:

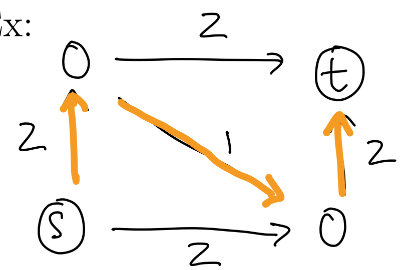


Notice that above there are no more paths where we can push more flow. However the max flow is not 16. It is 19!!

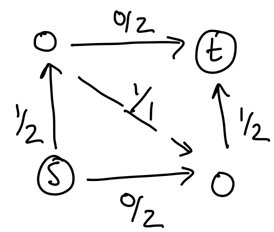
The problem:

By selecting paths like this, we have no way to *undo* a decision that might have been the wrong one...

Ex:



If we pick the yellow path first, then we will never find the max flow $\Longrightarrow$

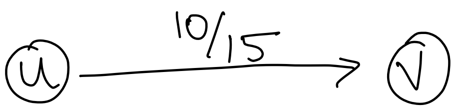


eventually the flow will 3, but the max flow is 4.

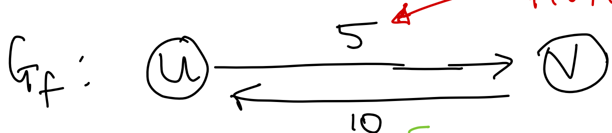
Solution:

We need to develop a method to *remove* flow.

Suppose we have the edge in G:



Build a **Residual Network**, G_f with edges as follows:



$C_f(u,v)$ = Residual capacity.
$= c(u,v) - f(u,v) = 15 - 10 = 5.$

$C_f(v,u) = 10.$
could "push back" the 10 units...

Note that $|E_f| \leq 2|E|$.

Another example of an edge in G and G_f:

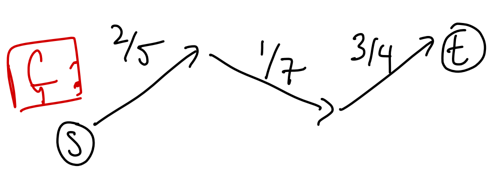
$C_f(u,v) = 10-7$

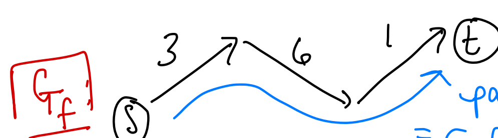
$C_f(v,u) = c(v,u) + f(u,v) = 7+1 = 8.$

The FORD FULKERSON METHOD:: For finding maximum flows.

- Similar to the above greedy approach, but will look for paths in the **residual network**.
- Start with all flows $f(u,v) = 0$.
- While there exists an augmenting path from s to t in G_f, identify the *capacity* of that path. Let $f^\star$ be the flow in G_f along this path.

ex





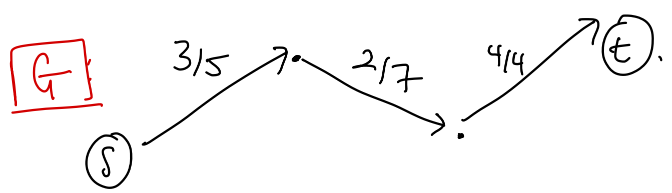
path capacity

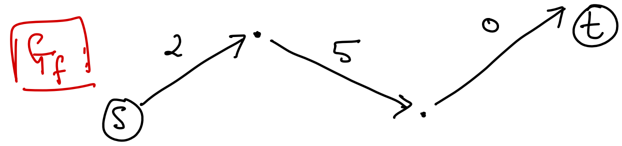
$= C_f(p)$

$= \min_{(u,v) \in P} C_f(u,v) = 1.$

Augment the flow in G by $f^{\star}$.

Above, $|f^*| = 1$

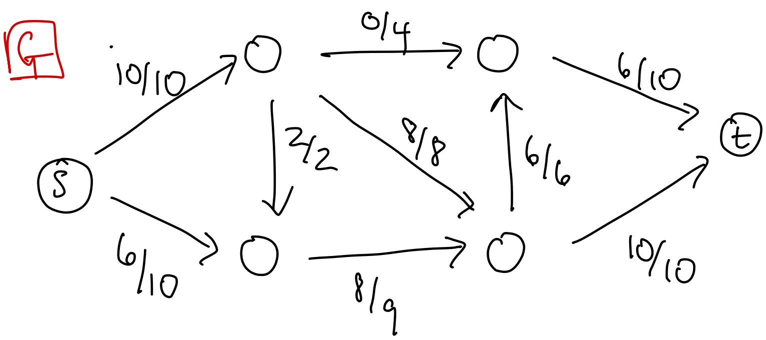




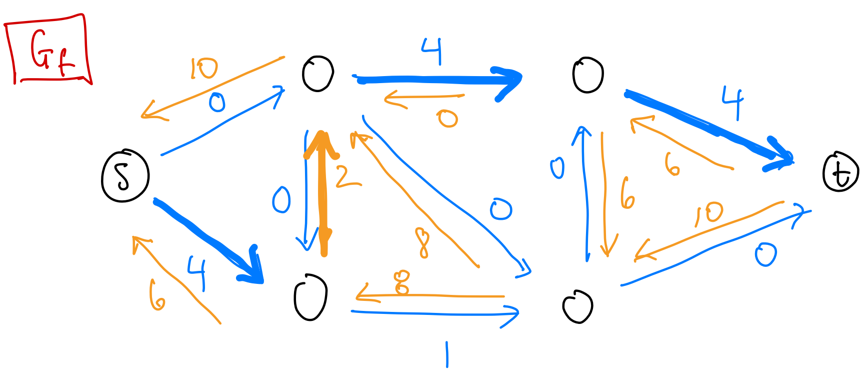
The flow in G is increased by the amount $|f^*|$.

- Continue until there are no more paths in G_f from s to t.

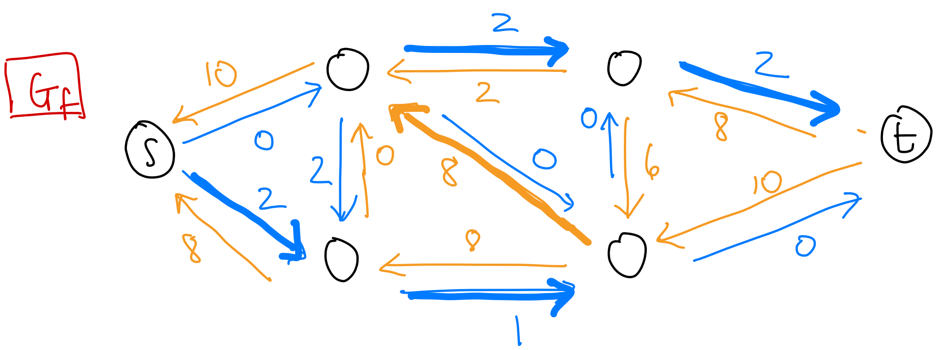
Let's look at the previous example where we were unable to find the maximum flow: .



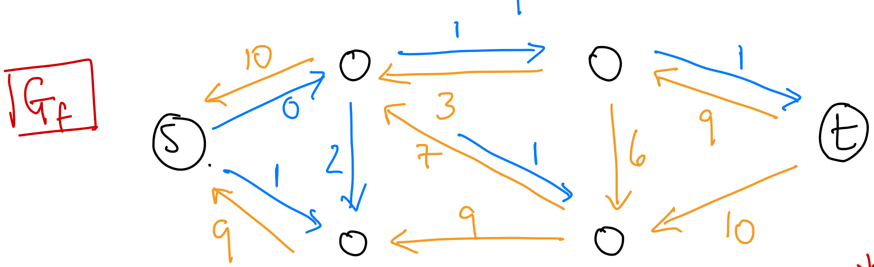
flow was 16. (not yet max...)



aug. path has capacity 2.



aug. path has capacity 1.



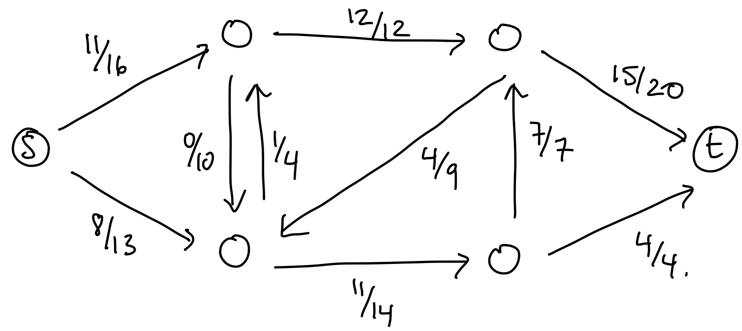
Final flow $= 16 + 2 + 1$

$= 9.$

* no more paths!

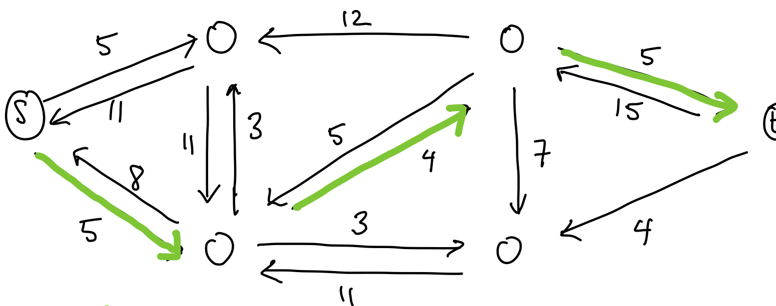
Here is an example with a double edge:

G



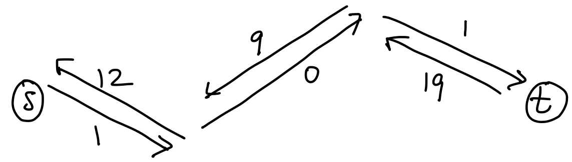
flow = 19.

G_f



$c_f(p) = 4$. So new flow = 19 + 4 = 23.

aug. by 4 !!



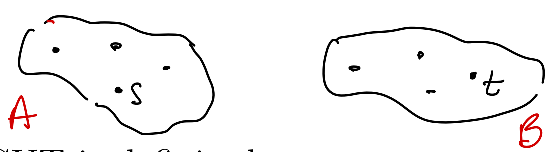
c_f values in G_f paths after augmentation by 4.

How do we find the augmenting paths:

Notice that this is just a path in the graph G_f from s to t. Any traversal algorithm that can search from s to t will work. Ex. DFS.

Cuts in the Network:

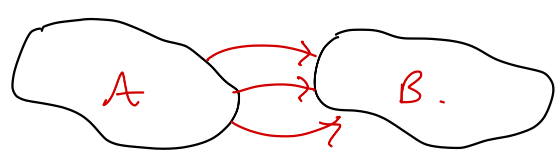
A CUT is a partition (A, B) of the vertices V such that $s \in A$, $t \in B$.



$\boxed{A \cup B = V.}$ (don't leave out vertices!)

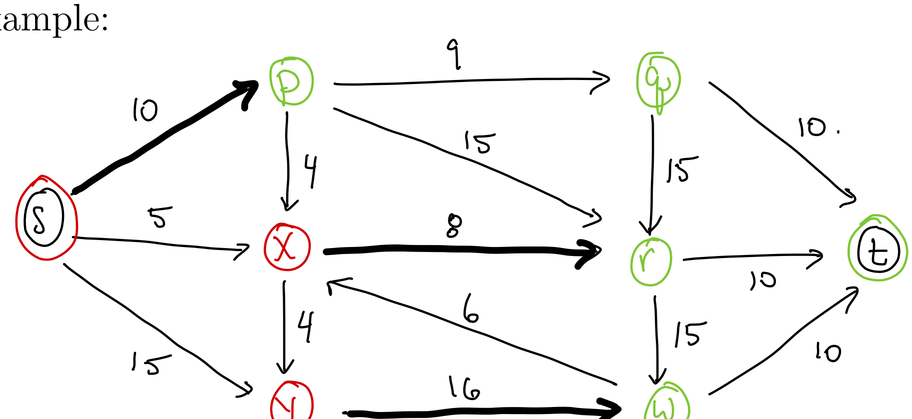
The capacity of the CUT is definited as :

$$cap(A,B) = \sum_{\substack{u \in A \\ v \in B}} c(u,v)$$



Sum these capacities on the edges $A \to B$.

Example:



$A = s, x, y$ $\quad B = p, q, r, w, t.$

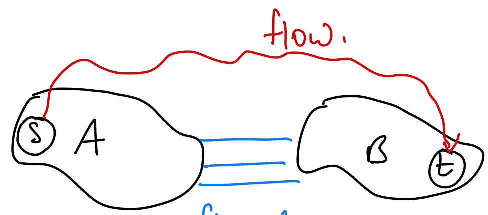
$$cap(A,B) = \text{edges } A \to B$$
$$= 10 + 8 + 16$$
$$= 34.$$

In the above example, there is another cut of size 28. It is in fact the minimum of all cuts.

Relationship between Cuts and flows:

1. If f is any flow, and (A, B) is any cut, then:

$$val(f) = f(A, B)$$



flow from $s \to t$ must pass through these edges.

Pf: $val(f) = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$ (defn)

$$= \sum_{u \in A} \Big[\underbrace{\sum_{v \in V} f(u,v) - \sum_{v \in V} f(v,u)} \Big]$$

this will be 0 for all u except $u = s$

$$= \sum_{u \in A} \sum_{v \in B} f(u,v) + \sum_{u \in A} \sum_{v \notin B} f(u,v) - \sum_{u \in A} \sum_{v \in B} f(v,u) - \sum_{u \in A} \sum_{v \notin B} f(v,u)$$

together this makes $v \in V$.

$= f(A,B)$ $\qquad$ $= 0$.

2. If f is any flow, then its value is bounded by the capacity of *any* cut.

$$\boxed{val(f) \le cap(A,B)}$$

Pf: $val(f) = \sum_{u \in A} \sum_{v \in B} f(u,v) - \sum_{u \in A} \sum_{v \in B} f(v,u)$

$$\le \sum_{u \in A} \sum_{v \in B} f(u,v) \le \sum_{u \in A} \sum_{v \in B} c(u,v) = cap(A,B)$$

3. If for some flow f and some (A, B) cut,

$$val(f) = cap(A, B)$$

then f is a maximum flow and the cut is the minimum of all cuts.

MAX-FLOW MIN-CUT.

Pf: Assume f' is some other flow.

$val(f') \leq cap(A,B)$ by prop. 2

$= val(f)$

So f is max. flow!

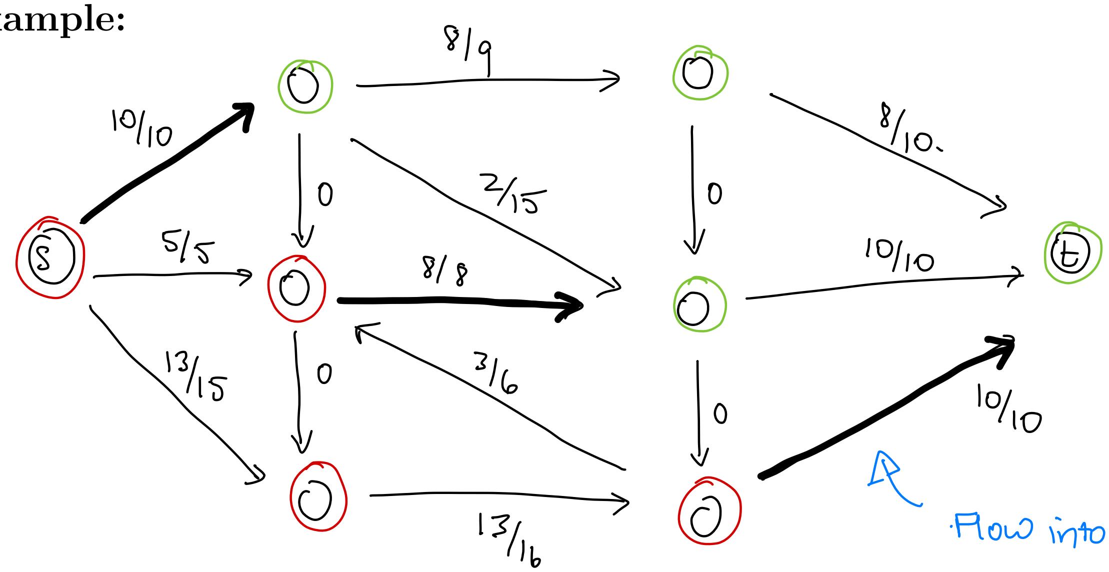
Assume (A', B') is some other cut...

Then $cap(A', B') \geq val(f)$ by prop 2.

$= cap(A,B)$

So (A,B) is the minimum cut.

Example:



Flow into t is 28.

A = set of red.

$cap(A,B) =$

$10 + 10 + 8 = 28.$

This is the min cut, and so the max flow is also 28.

The above properties will be used to prove the following theorem:

Theorem: A flow f is maximal if there are no augmenting paths in the residual network.

We shall prove that the following are equivalent:

- $cap(A, B) = val(f)$ 1
- $\Leftrightarrow$ f is a maximum flow 2
- $\Leftrightarrow$ No augmenting paths in G_f 3

Pf:

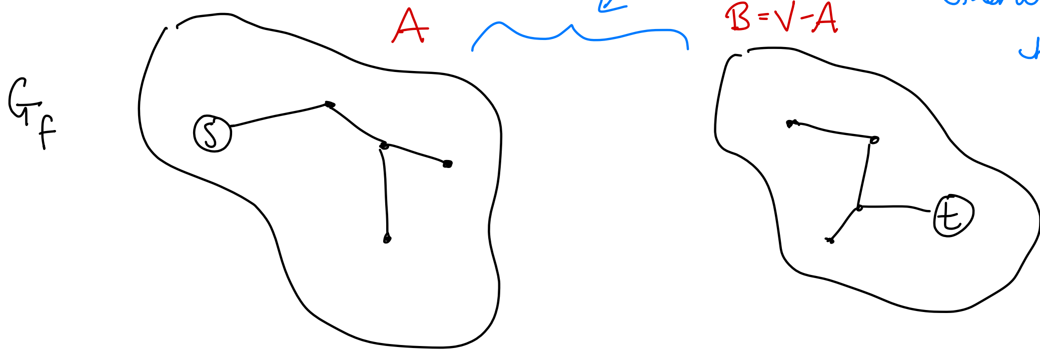
1 $\Rightarrow$ 2 max flow, min cut theorem. ✓

2 $\Rightarrow$ 3 if there were more augmenting paths, then the flow would not have been optimal.

3 $\Rightarrow$ 1. Assume f has no aug. paths in G_f. (ie: no $s \to t$ paths).

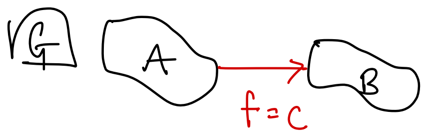
Then let A be the vertices we can reach from s, in the graph G_f:

no edges $A \to B$ in G_f otherwise there would have been a path $s \to t$.



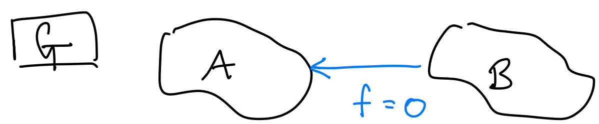
$$\mathrm{val}(f) = \underbrace{\sum_{\substack{u \in A \\ v \in B}} f(u,v)}_{} - \underbrace{\sum_{\substack{u \in A \\ v \in B}} f(v,u)}_{}.$$

The flow over an edge $A \to B$ in G must equal its capacity!



otherwise this edge would exist in $\underline{\underline{G_f}}$.

The flow over an edge $B \to A$ in G must be 0, otherwise there would be a $A \to B$ edge in G_f.

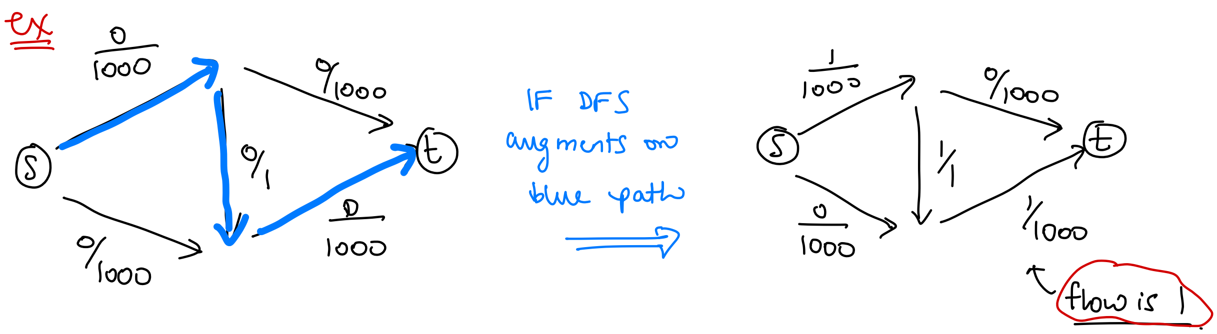


$$\therefore \ \mathrm{val}(f) = \sum_{\substack{u \in A \\ v \in B}} c(u,v) - 0 = \mathrm{cap}(A,B).$$

Complexity:

- Note that $|V|$ is $O(|E|)$ for these connected graphs.
- Algorithm repeatedly performs DFS until there are no more aug. paths.
- The time to update G_f after each iteration is $O(|E|)$.
- Thus we need to bound the # of iterations.

- Graph traversal (DFS) and edge updates take $O(|E|)$.
- The number of iterations can be as much as the value of f:



- Clearly the max flow is 2000
- it could be reached in 2 iterations!

It will take 1999 more iterations using these paths until we find max flow!

- Total: $O(|E| \cdot Val(f))$

The complexity above assumes integer capacities. If the capacities are *rational* the algorithm is guaranteed to finish, however for some irrational capacities, it is non-terminating.

There are several methods to find *better* augmenting paths...

Edmonds-Karp algorithm ('72) referrs to using BFS instead of DFS to find the augmenting paths. In this case, each BFS takes $O(|E|)$ time, as with DFS, however the number of iterations is bounded by $O(|V \cdot E|)$.

- Total complexity: $O(|V| \cdot |E^2|)$
- The idea here is that by using BFS to find the shortest paths $s \to t$, we can bound the # of times a path is used in the algorithm.
- There are many other possible ways of picking "good" augmenting paths.