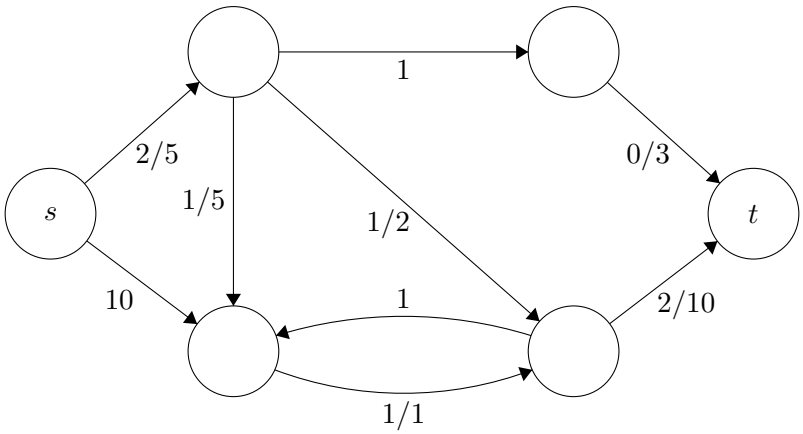
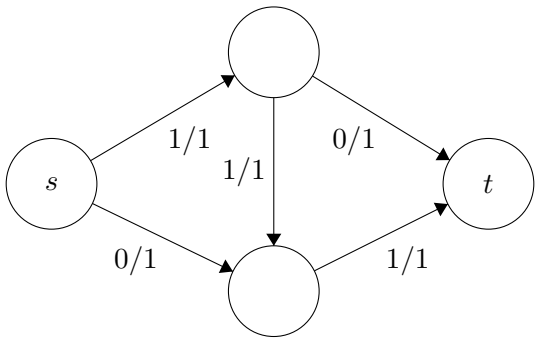
Review from the first lecture, and reminder of definitions
A flow netwrok:



$val(f) = f^{out}(s)$, we want to maximize this. as we can see, we still can increase the the flow coming out of the source.

Here is a naive way to try and solve it:
Try to find an s-t path that we still can increase its capacity until there is none.
However, There is an issue here, here is an example:



the naive algorithm found that path first, however it is not optimal.

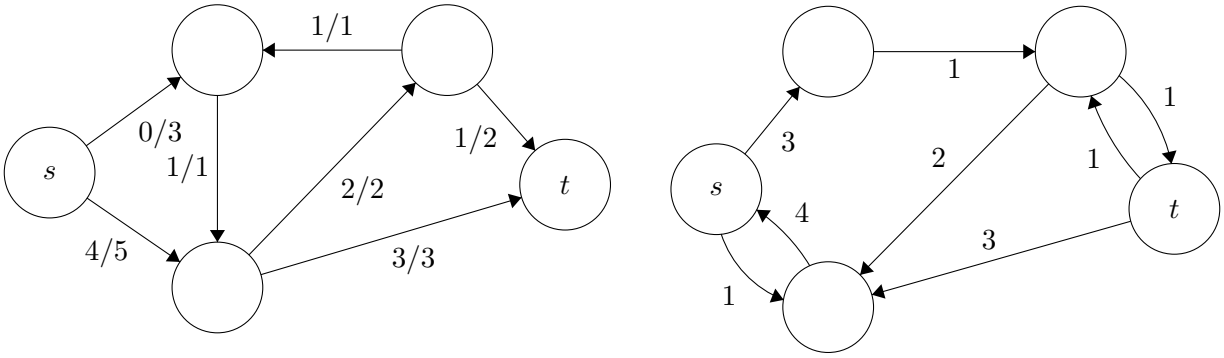
Ford-Fulkerson Algorithm:

1. start from all zero flows
2. Find a path from s-t s.t the edges that are in forward directions have unused capacities and the backward edges have a strictly positive flow on them, add one unit to forward edge and subtract one from backward edge

Def [Residual graph]:
Given a flow network (G, s, t, $\{C_e\}$) and a flow f on G, the residual graph G_f is as follows:

1. Nodes are the same as G.
2. For every edge $uv \in G$ with $f(uv) < C_{uv}$, add the edge uv with residual capacity $C_{uv} - f(uv)$ to G_f.
3. For every $uv \in G$ with $f(uv) > 0$, add the opposite edge vu with residual capacity f(uv)

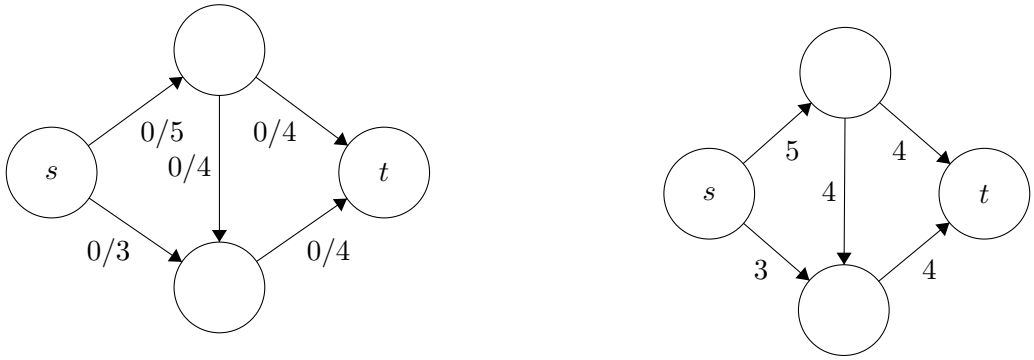
Here is an example of a graph and its residual graph:



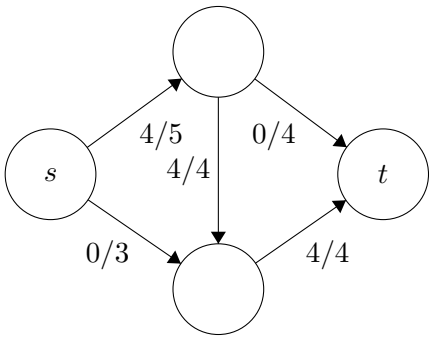
Ford-Fulkerson
- initially set $f(e) = 0$.
- while there is an s-t path P in G_f :
—- f' ¡- augment (f,p)
—- update f¡-f'
—- update G_f
end while.

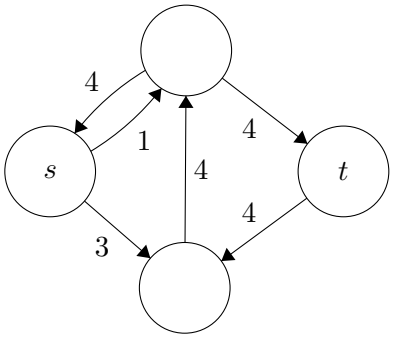
Augment(f,p): find the bottleneck α, which is the smallest capacity on P. For forward edge, add α to the flow, For backward edges, subtract α

here is an application:

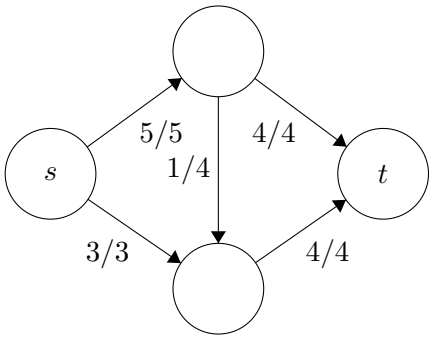


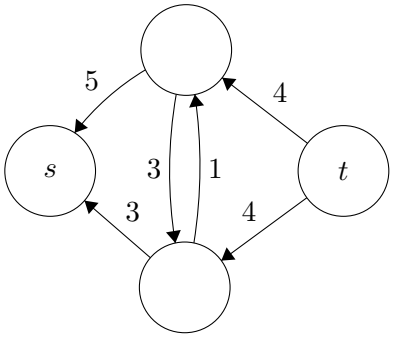
we find the first path with bottleneck equal to 4 and we get the following





this keeps going on till we have no paths from s to t and we get the following:





Claim: Ford-Fulkerson always return a valid flow.
Proof:

- Residual capacities are chosen so that updating with augment(f,p) will never assign a number to an edge that is larger than its capacities or smaller than 0. $\implies$ capacity condition is satisfied throughout the algorithm
- now for the conservation condtion: suppose the bottleneck on a path is α,
  1. for any forward edge, $f^{in} := f^{in} + \alpha$ and $f^{out} := f^{out} + \alpha$.
  2. $f^{in} := f^{in} + \alpha - \alpha$ and $f^{out} := f^{out}$
  3. $f^{in} := f^{in}$ and $f^{out} := f^{out} + \alpha - \alpha$
  4. $f^{in} := f^{in} - \alpha$ and $f^{out} := f^{out} - \alpha$

Hence the flow remains valid.

Claim: The algorithm terminates. **Proof:**
At every iteration, the flow increases by at least one unit, it can never exceed the total sum of all the capacities so it has to terminates at some point.
Running time: Let K be the largest capacity, n the number of vertices and m number of edges. We have at most Km iterations, each iteration requires a DFS in the residual grpah and update which is O(m+n) = O(m) because we assumed that every vertex is adjacent to at least one edge ($n/2 < m$). Hence the total running time is $O(Kmxm) = O(km^2)$

Def[s-t cut]:
A cut in a flow network is a partition (A,B) of the vertices such tat $s \in A,\ t \in B$.

Def[Cut capacity]:
It is the sum of the capacities going from A to B.

$$cap(A, B) = \Sigma_{uv \in E, u \in A, v \in B} C_{uv}$$

a graph with n vertices has 2^{n-2} (s,t) cuts