Theorem 1. Suppose $f$ is a feasible $s,t$ flow $f$, and $S$ is an $s,t$ cut $S$ such that

1. $f(e) = u(e)$ for all $e \in \partial^+ S$
2. $f(e) = 0$ for all $e \in \partial^- S$

Then $f$ is a maximum $s,t$ flow, $S$ is a minimum $s,t$ cut, and their values are the same.

In this lecture, we will prove the strong duality theorem: in any network, the maximum value of an $s,t$ flow equals the capacity of the minimum $s,t$ cut. We do so via an algorithm. That is, we describe an algorithm which in one swoop solves both the MAX-$s,t$-FLOW and the MIN-$s,t$-CUT problem, and also proves their respective values are the same. This algorithm was designed by Lester Ford and Dilbert Ray Fulkerson in the 1950s, and is called the Ford-Fulkerson algorithm. To describe this, we first introduce the concept of the residual networks.

1 The Residual Network

Let us start with an algorithm for finding maximum flows that doesn’t work. Recall what we need to do: we need to find a valid flow $f : E \to \mathbb{R}_{\geq 0}$ such that $\text{excess}_f(t)$ is maximized. We start with the zero flow: $f(e) = 0$ for all $e \in E$, and try to increase this flow in iterations. Now consider an $s,t$-path $p$ in the graph $G$. Given such a path $p$, we can augment the current flow $f$ along the path $p$ as follows:

- Let $\delta = \min_{e \in p} u(e)$
- For every $e \in p$, set $f(e) \leftarrow f(e) + \delta$.

Note that flow conservation remains valid; the total in-flow at any $v \neq s,t$ is equal to the total out-flow – it is either $\delta$ or 0. Also note that by choice of $\delta$ and since we started from the 0-flow, the capacity constraint also remains valid. Finally, the $\text{excess}_f(t)$ increases by $\delta$. Progress!

How should we proceed? We could repeat the steps above, namely, find another $s,t$-path $p$ and then augment flow along path $p$. However, we have already sent some flow which could have used up some capacity of certain edges $e$. In the augmentation step we should be wary of this lest we violate the capacity constraint. The fix is to maintain a residual capacity $uf(e)$ for every edge $e$. These are initially set to $u(e)$, the original capacity, but for every unit of flow that we pass through this edge, we must decrease its residual capacity. This leads to the following augmentation procedure along path $p$ given we have sent flow $f$:

- Let $\delta = \min_{e \in p} uf(e)$
- For every $e \in p$, set $f(e) \leftarrow f(e) + \delta$. 

In the last lecture we showed that the maximum $s,t$ flow is at most the minimum $s,t$ cut. Furthermore we looked at the conditions which would prove max-flow equals min-cut. Let’s recall that theorem for we will use this later.

In this lecture, we will prove the strong duality theorem: in any network, the maximum value of an $s,t$ flow equals the capacity of the minimum $s,t$ cut. We do so via an algorithm. That is, we describe an algorithm which in one swoop solves both the MAX-$s,t$-FLOW and the MIN-$s,t$-CUT problem, and also proves their respective values are the same. This algorithm was designed by Lester Ford and Dilbert Ray Fulkerson in the 1950s, and is called the Ford-Fulkerson algorithm. To describe this, we first introduce the concept of the residual networks.
• For every $e \in p$, set $u_f(e) \leftarrow u_f(e) - \delta$.

The above process can be repeated over and over again, and every time the value of the flow increases by $\delta$. We stop when $\delta = 0$, that is, we can’t find any path $p$ from $s$ to $t$ with $\min_{e \in p} u_f(e) > 0$. How would we check this? Simple: remove all edges with $u_f(e) = 0$ and check if there is a path from $s$ to $t$. We write the full algorithm below.

```
1: procedure NAIVEMAXFLOW(G, s, t, u):
2:     Start with $f \equiv 0$ and $u_f(e) = u(e)$ for all $e$.
3:         ▷ Invariant: $u_f(e) + f(e) = u(e)$ for all $e$.
4:     while true do:
5:         Find any path $p$ from $s$ to $t$ with $\min_{e \in p} u_f(e) =: \delta > 0$.
6:         If no such path break
7:         For every edge $e \in p$: $f(e) \leftarrow f(e) + \delta$; $u_f(e) \leftarrow u_f(e) - \delta$.
8:     return $f$
```

As can be guessed by the name and the color of the shading, the algorithm above, although a solid try, doesn’t return the correct solution. Let’s see an example where it fails (maybe you’d like to try to find one first before peeking?): see Figure 1.

![Figure 1](image_url)

Figure 1: In this graph $G$, all edges have unit capacity. If we send our first augmentation along the path $p = (s, a, b, t)$, then we would send 1 unit of flow on this. All these edges would have $u_f(e) = 0$ and deleting these edges disconnects $s$ and $t$. Thus the NAIVEMF algorithm would terminate. On the other hand, there is a flow of value 2 which sets $f(e) = 1$ for all edges except $(a, b)$. This would have been achieved if we sent flow first on the path $(s, x, b, t)$ and then $(s, a, y, t)$. But how would we know to do that?

In a sense, the flow we chose to send, that is the one on the path $(s, a, b, t)$ was a mistake. The main idea behind the notion of the residual network is to keep safeguards which help us correct mistakes when made. This is a general life principle, but something which beautifully works in the case of $s, t$ flows.

**Definition 1.** Given a flow network $(G, s, t, u)$ and a valid flow $f : E \rightarrow \mathbb{R}_{\geq 0}$, the residual network with respect to flow $f$ denoted as $G_f$ is defined as follows:

• $G_f = (V, E_f)$ where $E_f = E \cup E_{rev}$
• $E_{\text{rev}} = \{(v, u) : f(u, v) > 0\}$, that is, $E_{\text{rev}}$ contains the reverse of all edges which carry positive flow.

• The residual capacity on edges in $E_f$ is defined as follows

$$u_f(x, y) = \begin{cases} u(x, y) - f(x, y) & \text{if } (x, y) \in E \\ f(y, x) & \text{if } (x, y) \in E_{\text{rev}} \end{cases}$$

Let us draw the reverse graph for the network in Figure 1 with respect to the flow of unit 1 sent along the path $s, a, b, t$. This is shown in Figure 2.

![Figure 2](image)

Figure 2: The graph in the left shows the flow in green. The graph in the right is the residual graph. The red edges are $E_{\text{rev}}$. The numbers are the residual capacities.

Why is the residual network important? Well, note that after the flow $f$ is sent on the path $(s, a, b, t)$, the residual network $G_f$ does have a path from $s$ to $t$ where every edge has a residual capacity $u_f(e) > 0$; this path is $q = (s, p, b, a, q, t)$. As you can see, this path contains one edge $(b, a) \in E$ but in $E_{\text{rev}}$.

The question that should come into your mind now is: so what? The edge $(b, a)$ doesn’t even exist in the graph $G$; why are we bothering with such abstract constructs? Well, suppose you suppressed those thoughts and tried to augment flow along this path $q$. (Wait! Firstly there is no edge $(b, a)$ and now you are asking me to send flow across it? ) But here’s the point: we know that since $(b, a) \in E_{\text{rev}}$ there must exist $(a, b) \in E$ with $f(a, b) > 0$. Indeed, $f(a, b) = u_f(b, a)$. So increasing flow along the dummy reverse edge $(b, a) \in E_{\text{rev}}$ is actually just a short-hand for decreasing the flow along the edge $(a, b)$. This augmentation is indicating that our first choice of sending flow across the edge $(a, b)$ was perhaps a “mistake”, and this is fixing it. Indeed, this is the conceptual abstraction of the residual network: send flow along edges, but keep the reverse edges as stop guards to rectify potential mistakes. Now we are ready to formally give the algorithm.

2 The Ford Fulkerson Algorithm

First, we formally define what augmentation along a path in a residual network means.
The following invariants should be checked from the pseudocode above.

**Claim 1 (Invariants of Augmentation).**

1. For every edge \( e \in E \cup E_{rev} \), \( u_f(e) \geq 0 \).
2. For every edge \( (x, y) \in E \), \( f(x, y) + u_f(x, y) = u(x, y) \).
3. For every \( (x, y) \in E_{rev} \), \( f(y, x) = u_f(x, y) \).

**Proof.** This follows from the Invariants: For any edge \( (x, y) \in E \), we have \( f(x, y) = u(x, y) - u_f(x, y) \leq u(x, y) \) (from I2 and I1, respectively). Similarly, I1 implies \( u_f(y, x) \geq 0 \), that is, \( f(x, y) \geq 0 \).

**Claim 2.** If \( f \) satisfied the capacity constraints before AUGMENT, then it does so after AUGMENT too.

**Proof.** If \( v \notin p \), then there is nothing to discuss. So assume \( v \in p \). Since \( v \notin \{s, t\} \) it is an internal node in \( p \) and let \( (w, v) \) and \( (v, x) \) be the two edges of \( p \) incident on it. There are four cases to consider.

- Case 1: \( (w, v) \in E, (v, x) \in E \). In this case, both \( f(w, v) \) and \( f(v, x) \) go up by \( \delta \), implying the increase in excess is 0.
- Case 2: \( (w, v) \in E, (v, x) \in E_{rev} \). In this case, \( f(w, v) \) goes up by \( \delta \) and \( f(v, x) \) goes down by \( \delta \), implying the increase in excess is 0.
- Case 3: \( (w, v) \in E_{rev}, (v, x) \in E \). In this case, \( f(v, w) \) goes down by \( \delta \) and \( f(v, x) \) goes up by \( \delta \), implying the increase in excess is 0.
- Case 4: \( (w, v) \in E_{rev}, (v, x) \in E_{rev} \). In this case, both \( f(v, w) \) and \( f(v, x) \) go down by \( \delta \), implying the increase in excess is 0.

Let \( (v, t) \in p \) be the edge incident on \( t \). If \( (v, t) \in E \), then \( f(v, t) \) increases by \( \delta \) and the flow on no other edge incident on \( t \) changes, implying \( excess_f(t) \) goes by \( \delta \). If \( (v, t) \in E_{rev} \), then \( f(t, v) \) decreases by \( \delta \) and the flow on no other edge incident on \( t \) changes, implying \( excess_f(t) \) goes by \( \delta \).

Now we are ready to describe the maximum flow algorithm.
1: procedure FORDFULKERSON\( (G, s, t, u) \):
2:   Initialize \( f \equiv 0 \) and \( u_f \equiv u \) and \( G_f \equiv G \).
3:   while true do:
4:     Check if there is an \( s, t \) path \( p \) in \( G_f \) with all \( u_f(e) = 0 \) edges removed.
5:       If not, break.
6:     Else, AUGMENT\( (G_f, s, t, p) \).
7:   return \( (f, G_f) \).

Lemma 1. If \( u(e) \)'s are integer valued, then FORDFULKERSON returns an integer valued valid \( f \) in \( O(nmU) \) time, where \( U := \max_{e \in E} u(e) \).

Proof. Since the 0-flow is valid, and the Augmentation Claims imply AUGMENT maintains validity, we get that the final flow is valid. We claim that the Line 4 in AUGMENT will set \( \delta \) to a positive integer valued. To see this, we need to prove \( u_f \) is integer valued. But this is true in the beginning (when \( u_f \equiv u \)), and since subsequently \( f \) is augmented in \( \delta \)-installments, the \( f \) is always integral which in turn leads to \( u_f \) being integral. Furthermore, each time \( \text{excess}_f(t) \) grows by \( \delta \geq 1 \). Since the final flow is valid, the total value of this flow \( \text{excess}_f(t) \leq nU \) since there can be at most \( n \) edges of the form \((v, t)\) and each has capacity at most \( U \). Thus, the algorithm terminates in \( O(nU) \) rounds. Finally, note each round takes \( O(n + m) \) time.

The next lemma proves the flow returned is a max-flow by showing that the conditions of Theorem 1 holds.

Lemma 2. The flow \( f \) returned by FORDFULKERSON when it terminates is a maximum valued flow.

Proof. We describe a cut induced by a subset \( S \) which satisfied the properties of the corollary. In fact, define

\[
S = \{v : v \text{ is reachable from } s \text{ in } G_f \text{ with all } u_f(e) = 0 \text{ edges removed.}\}
\]

Clearly, \( s \in S \). Since the algorithm terminates, \( t \notin S \).

Now fix an \((x, y)\) \( \in \partial^+(S) \). Since \( y \) is not reachable from \( s \) using positive residual capacity edges, we get \( u_f(x, y) = 0 \). By I2, this implies

\[
\text{For } (x, y) \in \partial^+(S), \quad f(x, y) = u(x, y)
\]

Now consider an \((x, y)\) \( \in \partial^-(S) \). Since \( x \) is not reachable from \( s \) using positive residual capacity edges, we get \( u_f(y, x) = 0 \) for \((y, x) \in E_{\text{rev}}\). That is,

\[
\text{For } (x, y) \in \partial^-(S), \quad f(x, y) = 0
\]

But these are precisely the conditions of Theorem 1. Thus, \( f \) is a maximum \( s, t \) flow and \( S \) is a minimum \( s, t \) cut. In one swoop, FORDFULKERSON (+ one DFS) finds not only the max-flow but also the min-cut.

Theorem 2. Given a flow network \((G, s, t, u)\) where \( u(e) \) is a positive integer for every edge \( e \in E(G) \), the FORDFULKERSON algorithm finds a maximum \( s, t \) flow which is integral, and a minimum \( s, t \) cut in \( O(nmU) \) time, and max \( s, t \) flow equals min \( s, t \) cut.