CLRS3 lemma 24.15 (used in Bellman-Ford). Consider a weighted directed graph. Consider any shortest path from $s$ to $t$,

$$s = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k = t.$$ 

Suppose we initialize the data structure by

- $v \cdot \text{dist} = \infty$ for all vertices other than $s$
- $s \cdot \text{dist} = 0$

and then we perform a sequence of relaxation steps that includes, in order, relaxing $v_0 \rightarrow v_1$, then $v_1 \rightarrow v_2$, then ... then $v_{k-1} \rightarrow v_k$. After these relaxations, and at all times thereafter, $v_k \cdot \text{dist} = \text{distance}(s \text{ to } v_k)$.

We'll prove by induction that, after the $i$th edge has been relaxed,

$$v_i \cdot \text{dist} = \text{distance}(s \text{ to } v_i)$$

**BASE CASE $i = 0$.** Note that $s = v_0$. We initialized $s \cdot \text{dist} = 0$, and distance($s$ to $s$) = 0, so the induction hypothesis is true.

**INDUCTION STEP:** ...
SECTION 6.1

Flow networks
THE FLOW PROBLEM
Consider a graph in which each edge has a capacity. How should we assign a flow to each edge, so as to maximize the flow value?
Methods of finding the minimum total kilometrage in cargo-transportation planning in space, A.N.Tolstoy, 1930

NOTICE: THIS DOCUMENT CONTAINS INFORMATION AFFECTING THE NATIONAL DEFENSE OF THE UNITED STATES WITHIN THE MEANING OF THE ESPIONAGE LAW, TITLE 18, U.S.C. SECTIONS 793 and 794. THE TRANSMISSION OR THE REVELATION OF ITS CONTENTS IN ANY MANNER TO AN UNAUTHORIZED PERSON IS PROHIBITED BY LAW.
Fig. 7 — Traffic pattern: entire network available

Legend:
- International boundary
- Railway operating division
- Capacity: 12 each way per day.
- Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction
- All capacities in \( \sqrt{1000} \)s of tons each way per day

Origins: Divisions 2, 3, 5, 6, 7, 8, 9 (Poland); B (Czechoslovakia); and 2, 3 (Austria)

Alternative destinations: Germany or East Germany

Note 11K at Division 9, Poland.
Fig. 7—Traffic pattern: entire network available

Legend:
- International boundary
- Railway operating division
- Capacity: 12 each way per day.
- Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction.

Origins:
- Divisions 2, 3W, 3C, 23, 13M, 13R, 12, 52 (USDA), and Romania
- Destinations: Divisions 3, 6, 9 (Poland); B (Czechoslovakia); and 2, 5 (Austria)
- Alternative destinations: Germany or East Germany

Note: 11K at Division 9, Poland

The Bottleneck
Given a directed graph with a **source vertex** \( s \) and a **sink vertex** \( t \), where each edge \( u \rightarrow v \) has a **capacity** \( c(u \rightarrow v) > 0 \), a **flow** \( f \) is a set of edge labels \( f(u \rightarrow v) \) such that

- \( 0 \leq f(u \rightarrow v) \leq c(u \rightarrow v) \) on every edge
- total flow in = total flow out, at all vertices other than \( s \) and \( t \)

and the **value of the flow** is

- \( \text{value}(f) = \text{net flow out of } s = \text{net flow into } t \)

**PROBLEM STATEMENT**

Find a flow with maximum possible value (called a **maximum flow**).
SECTION 6.2

Ford-Fulkerson algorithm
SIMPLE GREEDY STRATEGY

Look for a path to the sink along which we can increase flow, then increase it as much as we can. Repeat this, until we can’t reach the sink.
SIMPLE GREEDY STRATEGY

Look for a path to the sink along which we can increase flow, then increase it as much as we can. Repeat this, until we can’t reach the sink.
QUESTION. Can you find a larger-value flow than this?
Send some of your stuff to me, so I can siphon it off!

I’ll siphon some off here, from the a→b flow. Redirect some of your excess to t, so they don’t notice!

They’ve shown me I can increase my flow value!

Send some of your stuff to me, so I can siphon it off!
They've shown me I can increase my flow value!

I could extract a flow of 3 at b ...
They've shown me I can increase my flow value!

Or I could extract a flow of 3 at a...
They've shown me I can increase my flow value! I shall extract an extra flow of 2 at t.
Ford-Fulkerson algorithm

1. Start with zero flow
2. Run bandit search to discover if the flow to $t$ can be increased, and if so find an appropriate sequence of edges
3. If $t$ can be reached:
   update the flow along those edges, then go back to step 2
4. If $t$ can’t be reached: terminate.
STEP 2A. Build the residual graph, which has the same vertices as the flow network, and
- if $f(u \rightarrow v) < c(u \rightarrow v)$: give the residual graph an edge $u \rightarrow v$ with the label “increase flow $u \rightarrow v$”
- if $f(u \rightarrow v) > 0$: give the residual graph an edge $v \rightarrow u$ with the label “decrease flow $u \rightarrow v$”

STEP 2B. Look for a path from $s$ to $t$ in the residual graph. This is called an augmenting path.

STEP 3. Find an update amount $\delta > 0$ that can be applied to all the edges along the augmenting path. Apply it.
EXERCISE. Find a way to increase the flow value.
6. Graphs and subgraphs

6.1 Flow networks

6.2 Ford-Fulkerson algorithm

Lecture 17

Lecture 18

Lecture 19

6.1 Flow networks

6.2 Ford-Fulkerson algorithm

Algorithms tick: max-flow

Maximum flow with Ford-Fulkerson / Edmonds-Karp

In this tick you will build a Ford–Fulkerson implementation from scratch. In fact you will implement the Edmonds–Karp variant of Ford–Fulkerson, which uses breadth first search (BFS) to find augmenting paths, and which has $O(VE^2)$ running time.
The Integrality Lemma. If the capacities are all integers, then the algorithm terminates, and the resulting flow on each edge is an integer. The running time is $O(\text{val}(f^*) \times E)$ where $f^*$ is a max flow.
6. Graphs and subgraphs

6.1 Flow networks

6.2 Ford-Fulkerson algorithm

Lecture 17

Lecture 18

Lecture 19

6. Algorithms tick max-flow

Maximum flow with Ford-Fulkerson / Edmonds-Karp

In this tick you will build a Ford–Fulkerson implementation from scratch. In fact you will implement the Edmonds–Karp variant of Ford–Fulkerson, which uses breadth first search (BFS) to find augmenting paths, and which has $O(VE^2)$ running time.
Maximum Flow and Minimum-Cost Flow in Almost-Linear Time

Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva

We give an algorithm that computes exact maximum flows and minimum-cost flows on directed graphs with $m$ edges and polynomially bounded integral demands, costs, and capacities in $m^{1+o(1)}$ time. Our algorithm builds the flow through a sequence of $m^{1+o(1)}$ approximate undirected minimum-ratio cycles, each of which is computed and processed in amortized $m^{o(1)}$ time using a new dynamic graph data structure.

Our framework extends to algorithms running in $m^{1+o(1)}$ time for computing flows that minimize general edge-separable convex functions to high accuracy. This gives almost-linear time algorithms for several problems including entropy-regularized optimal transport, matrix scaling, $p$-norm flows, and $p$-norm isotonic regression on arbitrary directed acyclic graphs.