Recursive formula for DP

\( \text{OPT}(i, v) = \text{shortest path } P \text{ from } v \text{ to } t \text{ using } \textit{at most } i \text{ edges} \)

case 1: \( P \) contains at most \( i-1 \) edges
  - best path of this type has cost \( \text{OPT}(i-1, v) \)

case 2: \( P \) uses exactly \( i \) edges
  - for every neighbor \( w \) of \( v \), the min cost path through \( w \) has cost \( c(v, w) + \text{OPT}(i-1, w) \)
  - take the min over the out-neighbors of \( v \)

\[
\text{OPT}(i, v) = \begin{cases} 
0 & i = 0 \text{ and } v = t \\
+\infty & i = 0 \text{ and } v \neq t \\
\min \{ \text{OPT}(i - 1, v); \min_{w \in G[v]} c(v, w) + \text{OPT}(i - 1, w) \} & \text{otherwise}
\end{cases}
\]
example

source = A, destination t = C

\[ \text{OPT}(i, v) = \begin{cases} 
0 & \text{if } i = 0 \text{ and } v = t \\
+\infty & \text{if } i = 0 \text{ and } v \neq t \\
\min \{ \text{OPT}(i - 1, v); \min_{w \in G[v]} c(v, w) + \text{OPT}(i - 1, w) \} & \text{otherwise}
\end{cases} \]
Algorithm 1: Bellman-Ford\((G, c, t)\)

\[
\begin{align*}
1 & \quad S \leftarrow \text{length-n array /* keeps track of first edge on path */} \\
2 & \quad M \leftarrow (n) \times (n) \text{ array; /}
3 & \quad \text{/* set border cases */} \\
4 & \quad M[0, t] \leftarrow 0; \quad \text{\textit{Memo}} \quad \text{Successors} \\
5 & \quad S[t] \leftarrow \text{None}; \\
6 & \quad \text{for } v \text{ in } V - t \text{ do } \quad \Theta(n) \quad \Theta(m) = \Theta(|E|) \\
7 & \quad \quad \quad M[0, v] \leftarrow \infty; \quad \Theta(n) \\
8 & \quad \quad \quad S[v] \leftarrow \text{None}; \quad \Theta(n) \\
9 & \quad \text{for } i = 1 \ldots n - 1 \text{ do } \quad \Theta(n) \quad \Theta(nm) \\
10 & \quad \quad \text{for } v \text{ in } V \text{ do } \quad \Theta(n) \quad \Theta(nm) \\
11 & \quad \quad \quad \text{/* apply recursive formula */} \\
12 & \quad \quad \quad M[i, v] \leftarrow M[i - 1, v]; \\
13 & \quad \quad \quad \text{for } w \text{ in } G[v] \text{ do } \quad \Theta(nm) \quad \Theta(nm) \\
14 & \quad \quad \quad \quad \quad M \leftarrow \min\{M[i, v]; c[v, w] + M[i - 1, w]\}; \\
15 & \quad \quad \quad \quad \quad \text{if } M[i, v] \text{ s updated then } \\
16 & \quad \quad \quad \quad \quad \quad S[v] \leftarrow w; \\
17 & \quad \quad \quad \text{/* array of least cost path length */} \\
18 & \quad \text{return } M[n - 1, :], S;
\end{align*}
\]

time complexity: \(\Theta(nm)\)

space complexity: \(\Theta(n^2)\)
Bellman-Ford - backtracking — TopHat

Algorithm 1: B-FBacktrack(S, s, t)

/* S successor list, source s, destination t */
P ← empty list /* path */
return P

Fill the missing code!

A. for v in V do
   | P.append(S[v]);

B. for v in V do
   | if S[v] == t then
     | P.append(S[v]);

C. parent = { }/* hash table
for v in S do
   | parent[S[v]] = v;
for v in parent do
   | P.append(parent[v]);

D. v ← s;
while v ≠ t do
   | P.append(S[v]);
   | v ← S[v];
Bellman-Ford - backtracking

Algorithm 1: B-FBacktrack($S, s, t$)

```plaintext
/* $S$ successor list, source $s$, destination $t$ */
P ← [s] path;
v ← s;
while v ≠ t do
    v ← $S[v]$/∗ update to next hop ∗/
    P.append(v)
return P;
```

# of iterations = number of edges in the shortest path

time complexity: $O(n)$
A flow network is a tuple $G = (V, E, s, t, c)$.

- Directed graph $(V, E)$ with source $s \in V$ and sink $t \in V$.
- Non-negative capacity $c(e)$ for each $e \in E$.

**Intuition.** Abstraction for material flowing through the edges of a graph
- material originates at source and is sent to sink.
- capacity is the throughput of each edge
Network flow - intuition

A flow network is a tuple $G = (V, E, s, t, c)$.

- Directed graph (digraph) $(V, E)$ with source $s \in V$ and sink $t \in V$.
- Non-negative capacity $c(e)$ for each $e \in E$.

**st-flow.** A function $f$ on the edges representing the “amount of matter” through each edge.

$f(a, b) = 9$ means that 3 unit of flow material is being sent along edge $(a, b)$. 

[Diagram of a network flow graph with node labels and edge capacities]
Network flow - intuition

A flow network is a tuple $G = (V, E, s, t, c)$.

- Directed graph (digraph) $(V, E)$ with source $s \in V$ and sink $t \in V$.
- Non-negative capacity $c(e)$ for each $e \in E$.

st-flow. A function $f$ on the edges representing the “amount of matter” through each edge.

properties:
- the capacity is an upper-bound on the flow for each edge
- nodes don’t have any “storage capacity”
Network flow - intuition

A flow network is a tuple $G = (V, E, s, t, c)$.
- Directed graph (digraph) $(V, E)$ with source $s \in V$ and sink $t \in V$.
- Non-negative capacity $c(e)$ for each $e \in E$.

**st-flow.** A function $f$ on the edges representing the “amount of matter” through each edge.

Properties:
- upper bounded by the capacity on each edge
- for each node total amount flowing in = total amount flowing out (except s,t)
Network flow - intuition

A flow network is a tuple $G = (V, E, s, t, c)$.

- Directed graph (digraph) $(V, E)$ with source $s \in V$ and sink $t \in V$.
- Non-negative capacity $c(e)$ for each $e \in E$.

st-flow. A function $f$ on the edges representing the “amount of matter” through each edge.

properties:
- upper bounded by the capacity on each edge
- for each node total amount flowing in = total amount flowing out (except $s,t$)

\[
\begin{align*}
\text{flow} & \quad \text{capacity} \\
5/10 & \quad 4 & \quad 0/9 & \quad 5/15 & \quad 0/15 & \quad 0/10 \\
5/5 & \quad 0/15 & \quad 0/4 & \quad 5/8 & \quad 0/6 & \quad 5/15 & \quad 5/10 & \quad t \\
5/10 & \quad 0/16 & \quad v \\
5/15 & \quad 5/10 \\
\end{align*}
\]

\[
\begin{align*}
\text{in}(v) & = 5 + 5 + 0 \\
\text{out}(v) & = 5 + 5
\end{align*}
\]
**Network flow**

**Def.** An \( st \)-flow (flow) \( f \) is a function that satisfies:

- For each \( e \in E \) :
  \[
  0 \leq f(e) \leq c(e) \quad \text{[capacity]}
  \]

- For each \( v \in V - \{s, t\} \) :
  \[
  \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \quad \text{[flow conservation]}
  \]

analog: Kirchhoff’s first (current) law: For any node/junction in an electrical circuit, the sum of current flowing in is equal to the sum flowing out.

\[\text{flow } f(s, a) = 8 \]
\[\text{capacity } c(s, a) = 10\]

For simplicity we assume that there are no edges into \( s \) - it doesn’t change the generality of the explanation.
Def. An $st$-flow (flow) $f$ is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Def. The value of a flow $f$ is: $val(f) = \sum_{e \text{ out of } s} f(e)$
Network flow - TopHat

Def. An \textit{st-flow} (flow) \( f \) is a function that satisfies:

\begin{itemize}
  \item For each \( e \in E \) : \( 0 \leq f(e) \leq c(e) \) \quad \text{[capacity]}
  \item For each \( v \in V - \{s, t\} : \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \) \quad \text{[flow conservation]}
\end{itemize}

Def. The value of a flow \( f \) is: \( \text{val}(f) = \sum_{e \text{ out of } s} f(e) \)

What is the value of the flow in this graph?
A. 10  
B. 15  
C. 23  
D. 30

8 + 5 + 10 = 23
Network flow

Def. An \textit{st-flow (flow)} $f$ is a function that satisfies:

\begin{itemize}
  \item For each $e \in E$:
    \[ 0 \leq f(e) \leq c(e) \]  
    \textbf{[capacity]}
  \item For each $v \in V - \{s, t\}$:
    \[ \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \]  
    \textbf{[flow conservation]}
\end{itemize}

Def. The \textbf{value} of a flow $f$ is:
\[ \text{val}(f) = \sum_{e \text{ out of } s} f(e) \]

\[ \text{val}(f) = f_{\text{in}}(t) = 5 + 10 + 10 = 25 \]

\[ \text{val}(f) = f_{\text{out}}(s) = 10 + 5 + 10 = 25 \]
**Network flow**

**Def.** An *st*-flow (flow) $f$ is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: \[
\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)\] [flow conservation]

**Def.** The value of a flow $f$ is: $\text{val}(f) = \sum_{e \text{ out of } s} f(e)$

The values of $f$ in this picture depict a valid flow, but not the maximum amount that could be sent from $s$ to $t$.

\[
\begin{align*}
\text{val}(f) &= \text{f}_{in}(t) = 5 + 10 + 10 = 25 \\
\text{val}(f) &= \text{f}_{out}(s) = 10 + 5 + 10 = 25
\end{align*}
\]
Maximum-flow problem

Def. An \textit{st-flow (flow)} $f$ is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Def. The \textbf{value} of a flow $f$ is: $\text{val}(f) = \sum_{e \text{ out of } s} f(e)$

Max-Flow problem:

Given a directed graph with edge capacities, find the maximum value flow.

\[
\text{val}(f) = 10 + 5 + 13 = 28
\]
Integral maximum flow - TopHat

Integral flow: the flow on each edge is integer valued

Does this graph have an integer maximum flow?
A. yes
B. no

A network can have multiple max flows - the value of the maximum flow is **UNIQUE**
How to find a maximum flow?

Max-Flow problem:
Given a directed graph with edge capacities, find the maximum value flow.

“Find” flow = assign valid flow values to each edge.
  • valid = satisfies the capacity and flow conservation constraints

There is an efficient polynomial algorithm - Ford-Fulkerson
Augmenting paths

Observation: we can send additional flow along an st-path if there is free capacity along every edge.

Augmenting path: an st-path with free capacity, along which we augment the flow.
Augmenting paths

**Observation:** we can send additional flow along an st-path if there is free capacity along every edge.

**Augmenting path:** an st-path with free capacity, along which we augment the flow.

**GreedyFlow**\((G(V,E,c))\):

1. While there is an augmenting path \(P\):
2. increase flow along \(P\) by the max available
3. Return \(f\) along each edge
Augmenting paths

**Observation:** we can send additional flow along an st-path if there is free capacity along every edge.

**Augmenting path:** an st-path with free capacity, along which we augment the flow.

We can augment the flow by 8 units along the highlighted path.
Augmenting paths

**Observation:** we can send additional flow along an st-path if there is free capacity along every edge.

**Augmenting path:** an st-path with free capacity, along which we augment the flow.

We can augment the flow by 8 units along the highlighted path.
Augmenting paths

Observation: we can send additional flow along an st-path if there is free capacity along every edge.

Augmenting path: an st-path with free capacity, along which we augment the flow.
Augmenting paths

**Observation:** we can send additional flow along an st-path if there is free capacity along every edge.

**Augmenting path:** an st-path with free capacity, along which we augment the flow.
Augmenting paths and max flow

Value on top (the outcome of our greedy run) is 16
Below 19
Choosing augmenting path greedily

Greedy algorithm:
• start with $f(e) = 0$ on each edge
• find an st-path $P$ where each edge has $f(e) < c(e)$
• augment flow along path $p$
• repeat until you get stuck

max flow solution:

suboptimal due to greedy choice:
Choosing augmenting path greedily

Greedy algorithm:
- start with $f(e) = 0$ on each edge
- find an $st$-path $P$ where each edge has $f(e) < c(e)$
- augment flow along path $p$
- repeat until you get stuck

max flow solution:

Can we “push water back” on the middle edge to get from the greedy to the optimal solution?
Residual graph

G: nodes V, edges E, capacities c(e)

residual graph G_f:
- nodes V
- edges $E \cup E^{reverse}$
- residual capacity

$$c_f(e) = \begin{cases} 
  c(e) - f(e) & e \in E \\
  f(e) & e \in E^{reverse} 
\end{cases}$$

The sum of the capacity on an edge and its reverse are equal to the original edge capacity, thus

$$c_f(e) + c_f(e^{reverse}) = c(e)$$

- Initially the flow is 0 on every edge, hence the reverse edge capacities are 0
Residual graph

G: nodes V, edges E, capacities c(e)

residual graph G_f:
• nodes V
• edges \( E \cup E^{\text{reverse}} \)
• residual capacity

\[
c_f(e) = \begin{cases} 
  c(e) - f(e) & e \in E \\
  f(e) & e \in E^{\text{reverse}} 
\end{cases}
\]

The sum of the capacity on an edge and its reverse are equal to the original edge capacity, thus

\[
c_f(e) + c_f(e^{\text{reversed}}) = c(e)
\]

• Initially the flow is 0 on every edge, hence the reverse edge capacities are 0
Ford-Fulkerson algorithm

Algorithm:
• In each iteration find an augmenting path in $G_f$ and increase the flow along the path
  • Treat original and reversed edges the same in the paths
• For each residual edge capacity that is decrease along the path, increase the capacity of the opposite edge
  • maintain $c_f(e) + c_f(e^{\text{reversed}}) = c(e)$
Ford-Fulkerson algorithm

Algorithm:
• In each iteration find an augmenting path in $G_f$ and increase the flow along the path
  • Treat original and reversed edges the same in the paths
• For each residual edge capacity that is decrease along the path, increase the capacity of the opposite edge
  • maintain $c_f(e) + c_f(e_{reversed}) = c(e)$

send 1 unit of flow along the path s->u->v->t
• updated capacities in red
Ford-Fulkerson algorithm

Algorithm:
• In each iteration find an augmenting path in $G_f$ and increase the flow along the path
  • Treat original and reversed edges the same in the paths
• For each residual edge capacity that is decrease along the path, increase the capacity of the opposite edge
  • maintain $c_f(e) + c_f(e^{reversed}) = c(e)$
Ford-Fulkerson algorithm - TopHat

Algorithm:
• In each iteration find an augmenting path in $G_f$ and increase the flow along the path
  • Treat original and reversed edges the same in the paths
• For each residual edge capacity that is decrease along the path, increase the capacity of the opposite edge
  • maintain $c_f(e) + c_f(e^{reversed}) = c(e)$

Select all valid augmenting paths
A. $s \rightarrow u \rightarrow t$
B. $s \rightarrow u \rightarrow v \rightarrow t$
C. $s \rightarrow v \rightarrow t$
D. $s \rightarrow v \rightarrow u \rightarrow t$
Ford-Fulkerson algorithm

Algorithm:
• In each iteration find an augmenting path in $G_f$ and increase the flow along the path
  • Treat original and reversed edges the same in the paths
• For each residual edge capacity that is decrease along the path, increase the capacity of the opposite edge
  • maintain $c_f(e) + c_f(e^{reversed}) = c(e)$

![Diagram of a directed graph with labeled edges]

Value of the flow (in the original $G$) is the total residual capacity in $G_f$ leaving $t$. 
Ford-Fulkerson algorithm

• This is the situation where the greedy algorithm got stuck in a suboptimal solution
• However, there is an augmenting path using residual edges
• Using a residual edge is akin to “sending flow back” among an edge.
Ford-Fulkerson example

Residual graph $G_f$: (trace the example in class)
Ford-Fulkerson example

Residual graph $G_r$: 

Value of flow: 19
Ford - Fulkerson algorithm

Algorithm 1: FordFulkerson(G(V, E, c))

1. for e ∈ E do
2.     f(e) = 0;
3. G_f ← residual graph with respect to f;
4. while There exist an s ⇣ s augmenting path P do
5.     f ← Augment(f, c, P) /* augment the flow along the path with
        the bottleneck capacity along P */
6.     update G_f;
7. return f

What is the size of G_f?

\[ |G_r| = 2 |G| = 2 |E| \]

How do we find an augmenting path?

BFS/DFS with edges of non-zero capacity

How long does it take to augment a path? (i.e. run Augment(f, c, P) in line 5)

\[ O(n) \]

How many iterations of the While-loop?

Let C be the maximum flow = \( O(C) \)
Ford - Fulkerson algorithm

Algorithm 1: FordFulkerson($G(V, E, c)$)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>for $e \in E$ do</td>
</tr>
<tr>
<td>2</td>
<td>$f(e) = 0$;</td>
</tr>
<tr>
<td>3</td>
<td>$G_f \leftarrow$ residual graph with respect to $f$;</td>
</tr>
<tr>
<td>4</td>
<td>while There exist an $s \leadsto s$ augmenting path $P$ do</td>
</tr>
<tr>
<td>5</td>
<td>$f \leftarrow$ Augment($f, c, P$) /* augment the flow along the path with the bottleneck capacity along $P$ */</td>
</tr>
<tr>
<td>6</td>
<td>update $G_f$;</td>
</tr>
<tr>
<td>7</td>
<td>return $f$</td>
</tr>
</tbody>
</table>

What is the size of $G_f$?
- n nodes, 2m edges

How do we find an augmenting path?
- Run BFS from $S$ on the non-zero edges in $G_f$

How long does it take to augment a path? (i.e. run Augment($f, c, P$) in line 5)
- $O(n)$, since a simple path contains at most n edges

How many iterations of the While-loop?
- if all capacities are integers, then we can increase the flow by at least 1 in each iteration.
- the total sum of capacity $C = \sum_{e \in E} c(e)$ is an upper bound on the number of iterations
Ford - Fulkerson algorithm

Algorithm 1: FordFulkerson(G(V, E, c))

1. for e ∈ E do
   2. f(e) = 0;
   3. \( G_f \) ← residual graph with respect to \( f \);
4. while There exist an s \( \rightarrow \) s augmenting path \( P \) do
   5. \( f \) ← Augment\( (f, c, P) \) /* augment the flow along the path with the bottleneck capacity along \( P \) */
   6. update \( G_f \);
5. return \( f \)

Running time:
build \( G_f \): \( O(n+m) \)
one execution of the While loop:
   • \( O(n+m) \) to find the path with BFS + \( O(n) \) for augmenting the flow
Number of iterations: \( O(\text{val}(f)) \leq O(C) \) where \( C = \sum_{e \in E} c(e) \)
Total: \( O(mC) \)