Flow networks

How much flow can we push through from $s$ to $t$? (Numbers are capacities.)

max flow here is 3 units
Flow networks

How much flow can we push through from s to t?
(Numbers are capacities.)
Flow networks

How much flow can we push through from $s$ to $t$?  
(Numbers are capacities.)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Flow networks

How much flow can we push through from $s$ to $t$? (Numbers are capacities.)
Def:

A **flow network** is a directed graph \( G=(V,E) \) where edges have capacities \( c:E \rightarrow \mathbb{R}^+ \). There are two specified vertices \( s \) (source) and \( t \) (sink).

A **flow** \( f:E \rightarrow \mathbb{R} \) must satisfy:

- **Capacity constraint**: for every edge \( e \): \( f(e) \leq c(e) \)
- **Flow conservation**:
  
  for every \( v \) in \( V-\{s,t\} \):
  
  \[
  \sum_{e \text{ out of } v} f(e) = \sum_{e \text{ into } v} f(e)
  \]

The **value** of the flow is:

\[
\sum_{e \text{ out of } s} f(e)
\]
Maximum flow problem

Input:
a flow network $G=(V,E)$, with capacities $c$, the source $s$ and sink $t$

Output:
a maximum-value flow

Algorithm?
“Def”:

Given a flow $f$, an **augmenting path** is a path $s=v_1, v_2, ..., v_k=t$ such that

$$f(v_i,v_{i+1}) < c(v_i,v_{i+1}) \quad \text{for } i=1,\ldots,k-1$$

We can augment the flow along this path by

$$\min_{i \in [1,\ldots,k-1]} \{c(v_i,v_{i+1}) - f(v_i,v_{i+1})\}$$

---

**Ford-Fulkerson** ( $G=(V,E), c, s, t$ )

1. Initialize flow $f$ to 0
2. While exists augmenting path $p$ do
3. Augment flow $f$ along $p$
4. Return $f$
"Def": Given a flow $f$, an **augmenting path** is a path $s=v_1, v_2, ..., v_k=t$ such that

$$f(v_i,v_{i+1}) < c(v_i,v_{i+1}) \quad \text{for } i=1,\ldots,k-1$$

How to find augmenting paths?

*Ford-Fulkerson ( G=(V,E), c, s, t )*

1. Initialize flow $f$ to 0
2. While exists augmenting path $p$ do
3. Augment flow $f$ along $p$
4. Return $f$
"Def": Given is $G=(V,E), c, f$. The **residual graph** has edges weighted by the **residual capacities**, i.e. $c_f(e) = c(e) - f(e)$ if $c_f(e) > 0$.

**Ford-Fulkerson** ($G=(V,E), c, s, t$)
1. Initialize flow $f$ to 0
2. While exists augmenting path $p$ do
3. Augment flow $f$ along $p$
4. Return $f$
Maximum flow problem - Ford-Fulkerson

“Def”: Given is $G=(V,E), c, f$. The residual graph has edges weighted by the residual capacities, i.e. $c_f(e) = c(e)-f(e)$

Idea: Find an s-t path in the residual graph!

Ford-Fulkerson ( $G=(V,E), c, s, t$ )
1. Initialize flow $f$ to 0
2. While exists augmenting path $p$ do
3.  Augment flow $f$ along $p$
4. Return $f$
Consider this input:

Ford-Fulkerson \( (G=(V,E), c, s, t) \)
1. Initialize flow \( f \) to 0
2. While exists augmenting path \( p \) do
3. Augment flow \( f \) along \( p \)
4. Return \( f \)
Consider this input:

Need to refine the definition of augmenting paths and residual graph.

```
Ford-Fulkerson ( G=(V,E), c, s, t )
1. Initialize flow f to 0
2. While exists augmenting path p do
3.   Augment flow f along p
4. Return f
```
Refined def:
Given is $G=(V,E)$, $c$, $f$.
The residual graph $G_f=(V,E')$ contains the following edges:

- **forward edge:**
  if $e \in E$ and $f(e) < c(e)$ then include $e$ in $E'$ with weight
  $$c_f(e) = c(e)-f(e),$$

- **backward edge:**
  if $e=(u,v) \in E$ with $f(e)>0$ then include $(v,u)$ in $E'$ with weight
  $$c_f(v,u) = f(u,v).$$
**Ford-Fulkerson** (G=(V,E), c, s, t)

1. For every edge e let f(e)=0
2. Construct the residual graph G_f
3. While exists s-t path in G_f do
   4. Let p be an s-t path in G_f
   5. Let d=min_{e in p} c_f(e)
   6. For every e on p do
      7. If e is a forward edge then
         8. f(e)+=d
      else
         10. f(reverse(e))-=d
   11. Update G_f (construct new G_f)
12. Return f
Maximum flow problem - Ford-Fulkerson

Ford-Fulkerson ( G=(V,E), c, s, t )

1. For every edge e let f(e)=0 \{ O(m) \}
2. Construct the residual graph G_f \{ same as G \}
3. While exists s-t path in G_f do
4. Let p be an s-t path in G_f \{ O(n+m) \} e.g. BFS/DFS
5. Let d=\min_{e \in p} c_f(e) \{ O(n) \}
6. For every e on p do
7. If e is a forward edge then
\{ O(n) \}
8. f(e)+=d
9. else
10. f(reverse(e))-=d
11. Update G_f (construct new G_f) \{ O(n+m) \}
12. Return f

Running time:
Maximum flow problem – Ford-Fulkerson

This zig-zag path selection means: 2000 iterations.

For integer capacities,

\[ \text{# iterations} \leq \text{sum of edge capacities out of } s \]

The running time of FF: \( O(F(n,m)) \) = \( F \) (weakly polynomial)
Ford-Fulkerson (G=(V,E), c, s, t)
1. Initialize flow f to 0
2. While exists augmenting path p do
3. Augment flow f along p
4. Return f

Lemma:
Ford-Fulkerson works.

Let f be the flow returned by FF

Then, in its res. graph

Let A* be the set of vertices reachable from s in the final res. graph.

Observe: t & A* (bec. no s-t path)

no edges out of A*

A* is an s-t cut

edges into A*:
- if forward, then does not influence the value of the A* cut
- backward edges:
  fully saturated in the orig. graph,
  i.e., value of the flow is
  Sum of capacities of the reverse of these backward edges
  value of cut A* is the same

Flow value of the FF flow = cut value of A*

Since every flow value ≤ every cut value:

FF flow is max flow

A* is min cut
Lemma:
Ford-Fulkerson works.

Def:
Given $G=(V,E)$, c. An $s-t$ cut of $G$ is a subset of vertices $S$ s.t. $s \in S$ and $t \in \overline{S}$. Its value is

$$\sum_{e \text{ out of } S} c(e)$$
Lemma:
Ford-Fulkerson works.

The Max-flow min-cut theorem:
Let min-cut(G) be the minimal value of an s-t cut of G. Then:
\[ f \text{ is a maximum flow iff } \text{value}(f) = \text{min-cut}(G) \]

\[ \text{max-flow value} \leq \text{min-cut value} \leq \text{max-flow value} \]

\[ \text{easy to see because a flow from } s \text{ to } t \text{ needs to go through the cut edges, i.e. flow value} \leq \text{cut value} \]
Improving Ford-Fulkerson

Can find better paths to reduce the running time?

Ford-Fulkerson (G=(V,E), c, s, t)
1. Initialize flow f to 0
2. While exists augmenting path p do
3. Augment flow f along p
4. Return f
Improving Ford-Fulkerson

Can find better paths to reduce the running time?
- many ways, will discuss two:
  - Scaling paths
  - BFS

Ford-Fulkerson (G=(V,E), c, s, t)
1. Initialize flow f to 0
2. While exists augmenting path p do
3. Augment flow f along p
4. Return f
**Improving Ford-Fulkerson**

Can find better paths to reduce the running time?
- many ways, will discuss two:
  - Scaling paths

  for each $\Delta$, up to $2m$ augmenting paths, then go to $\frac{\Delta}{2}$
  overall log max capacity $\Delta$'s

  Overall running time: $O(m(n+m) \cdot \log \text{max cap})$

---

**Ford-Fulkerson (G=(V,E), c, s, t)**
1. Initialize flow $f$ to 0
2. While exists augmenting path $p$ do
3. Augment flow $f$ along $p$
4. Return $f$
Edmonds-Karp (G=(V,E), c, s, t)
1. Initialize flow f to 0
2. While exists augm. path p (check with BFS) do
3. Augment flow f along p
4. Return f

Can find better paths to reduce the running time?
- many ways, will discuss two:
  - Scaling paths
  - BFS

Thm: Edmonds-Karp takes $O(|V||E|)$ iterations.
Running time of Edmonds-Karp: $O(nm(n+m))$
Applications of Network Flows

- multiple sources, multiple sinks
Applications of Network Flows

- how to find minimum cut

![Network Flow Diagram]

- Example:
The $A^*$ set, i.e., vertices reachable from $s$ in the final residual graph.

Same running time as max flow.
Applications of Network Flows

- maximum number of edge-disjoint s-t paths
Applications of Network Flows

- maximum bipartite matching
Applications of Network Flows

- maximum weighted (perfect) bipartite matching
Consider the Diet problem:
- n food items, m nutrients
- for every nutrient: the daily quota $b_j$
- for each item: cost per pound $c_i$
- for every item and nutrient: how much of the nutrient in a pound of item: $a_{i,j}$

**Objective:** minimize the cost of feeding people while giving enough nutrients

**Variables:** $x_i$ → how much (in pounds) to buy of food item $i$

**Objective:** $\min \sum x_i c_i$

**Constraints:** for every nutrient $j$:
\[\sum x_i a_{i,j} \geq b_j\]

A linear program
A **linear program** looks like this:

Find $x_1, x_2, \ldots, x_m$ which

- maximize

$$c_1 x_1 + c_2 x_2 + \ldots + c_m x_m$$

- and satisfy these constraints:

$$a_{1,1} x_1 + a_{1,2} x_2 + \ldots + a_{1,m} x_m \leq b_1$$

$$a_{2,1} x_1 + a_{2,2} x_2 + \ldots + a_{2,m} x_m \leq b_2$$

$$\ldots$$

$$a_{n,1} x_1 + a_{n,2} x_2 + \ldots + a_{n,m} x_m \leq b_n$$
Introduction to Linear Programming

A **linear program** in compressed form:

Given a vector $c$ in $\mathbb{R}^m$, a vector $b$ in $\mathbb{R}^n$ and a matrix $A$ in $\mathbb{R}^{n \times m}$, find a vector $x$ in $\mathbb{R}^m$ which satisfies $xA^T \leq b$ and maximizes $cx^T$.

**Thm:**

Exists a polynomial-time algorithm solving linear programs.

**Caveat:**

Sometimes need **integer programs** (no algorithm for integer programs is likely to exist)!