**Max-Flow Algorithms**

Given $G, s, t$

**Ford Fulkerson**:
- While $E$ augmenting path on residual network $G$
  - push flow along path
- Running Time: $O(EE')$

**Edmonds-Karp**:
- F-F with BFS to find augmenting paths
  Running Time: $O(VE^2)$

---

Side note: push-relabel alg runs in $O(v^3)$ (used in practice?)
best known is $\approx O\left(\min(v^3, E^{2/3}E \cdot \log (v^{1/3}E))\log(\max\text{edge cost})\right)$

Many different variants of Max-flow can be solved with basic algorithm,
Example: Multi-source - Multi-sink $K$

Add "supersource" and "supersink," connect via edges w/infinite capacities

---

**Bipartite Matchings**

Defn: A Bipartite Graph $G$ is a graph whose vertices can be partitioned into $V=L \cup R$ st. all edges go between $L$ and $R$

Defn: A matching $M$ in $G$ is a subset of edges $M \subseteq E$ that touches each vertex at most once.
Goal: compute a **maximum bipartite matching** (matching of max cardinality)

Many applications (e.g., assigning students to recitations)

Solution:

- Make edges directed. Add s connected to \( V + e \), Add t connected from \( V + e \)
- All edges have capacity 1
- Max-flow solution is maximum matching. Can you see why?

Running time: \( O(VE^2) \) by Edmonds-Karp? Not quite

Actually \( O(VE) \) with Ford-Fulkerson regardless of BFS/DFS augmenting paths

- Reason is because flow increases by 1 with each augmentation
- Max-flow here is \( O(V) \)

Side notes: Better algorithm is Hopcraft-Karp \( O(VE) \), sketched in CLRS
Best known is \( O(V^w) \) where \( w = \) best known matrix mult exponent

(Mucha-Sankowski '04)

defn: a **perfect matching** is a matching that touches every vertex.

In particular, for bipartite \( G \) where \( |L| = |R| = n \), a perfect matching
is a maximum matching of size \( n \).

For unweighted graphs, finding a perfect matching is done by finding max matching.
But what if edges also have costs?

The **min-cost perfect matching** (aka "assignment") problem:

Find perfect matching of minimum cost.
Note: Problem is equivalent to \( \text{Max-cost(weight)} \) Perfect Matching Problem

To solve minimization problem, replace capacity \( c_e \) of each edge with \( (\text{max capacity}) - c_e \) and solve minimization problem.

Caution: Perfect is important here.

\[
\begin{array}{c}
0 & 1 & 5 \\
1 & 0 & 2 \\
5 & 2 & 0 \\
\end{array}
\]

Max-weight matching ≠ max-weight perfect matching

\[
\begin{array}{c}
1 & 0 \\
0 & -1 \\
1 & 0 \\
\end{array}
\]

\[\text{Min-cost Perfect Matching}\]

How to solve it? Many approaches

1. Linear Programming (will cover next week).
   Generic approach, polynomial-time but not very efficient for this problem.

2. Hungarian Method. Related to algorithm I'm going to sketch?

3. Variant of Ford-Fulkerson (from Kleinberg-Tardos text)
   With clever choice of augmenting paths,
   Let's sketch this alg.

Consider residual network \( G_f \) of some (partial) matching

\[
\begin{array}{c}
0 & 2 & 1 \\
3 & 0 & 4 \\
0 & 2 & 0 \\
\end{array}
\]

- "Backwards" Edges from \( R \) to \( L \) in \( G_f \) correspond to edges in current matching
- Forward edges have cost \( -c_e \). Makes sense to assign backwards edges cost \( -c_e \)
- Idea: Use minimum-cost augmenting paths
Still need to show

1) No negative cycles created

2) Algorithm terminates with min-cost assignment

Before showing (1) and (2), let's analyze running time

Running Time:
- \(O(n)\) iterations of shortest-path.
  - Using Bellman-Ford (deals with negative edge weights), total is \(O(n^2E)\)
  - Improvement allows use of Dijkstra, total is \(O(n^2 \log V + VE)\)

Correctness (Sketch):

Claim: \((1) \Rightarrow (2)\) Matching is min-cost iff no negative cycles in \(G_f\)

- If negative cycle exists, switch edge orientations to get cheaper matching

- If cheaper matching \(M'\) exists, take edges in symmetric difference of \(M, M'\)

These edges match same vertices, must form a set of directed cycle(s)

with cost \(= \text{cost}(M') - \text{cost}(M)\)

It remains to prove (1)

- Proof by induction. At each node \(v\), store \(d_M(v)\), min-cost distance from \(s\) to \(v\) under matching \(M\)

- Initially, for all edges \(e = (u, v)\), the quantity \(c_e + d(u) - d(v)\) is non-negative, and equal to zero for edges in \(M\)

- Use induction to show that for each successive addition to \(M\), this remains true for all edges \(e\)

- Thus for any cycle \(\Theta\), \(\sum_{e = (u, v) \in \Theta} (c_e + d(u) - d(v)) \geq 0\)

(see Kleinberg-Tardos for details)