Section 4.4: Algorithmic Matching

**Theorem 1:** Use Theorem 1, page 6 of Chapter 13 Notes, Math 605, Fall 2009

**Theorem 2:** Use Theorem 3, page 7 of Chapter 13 Notes

**Example 3:** Use Theorem 4, page 8 of Chapter 13 Notes

**Example 4:** Create network which keeps track of the remaining games and their outcomes. $X$ vertices: remaining teams. $Y$ vertices: Possible matchups between remaining teams. Flow into $X$ vertices: number of remaining games won. Flow into $Y$ vertices: number of games won by team with originating vertex. Flow out of $Y$ vertices: total number of games played by matchup represented by $Y$ vertices. Capacities: into $X$ should be amount to guarantee that Bears can tie this team. Out of $Y$ should be total number of games scheduled. $XY$ edges: no limit, so infinite capacity. There’s a solution to this problem if we can find a flow to saturate the edges out of $Y$.

Skip the optional material.

Section 4.5: The Transportation Problem

Set-up: Warehouses 1, 2, \ldots, $m$ with supplies $s_1, s_2, \ldots, s_m$.

Stores 1, 2, \ldots, $m$ with demands $d_1, d_2, \ldots, d_n$

Cost of shipping from warehouse $i$ to store $j$ is $c_{ij}$ dollars per unit.

Assume that the demand can be met. Find the minimal shipping cost. This requires total supply $\geq$ total demand. We can assume without loss of generality that total supply equals total demand by throwing in an additional store to demand the rest of the supply, with 0 shipping cost.

Model the shipment of $x_{i,j}$ units from warehouse $i$ to store $j$ by an edge from $i$ to $j$ weighted $x_{i,j}$ in the complete graph $K_{m,n}$. We want the sum of the edges out of vertex $i$ to equal $s_i$, and we want the sum of the edges into vertex $j$ to equal $d_i$. The total cost is represented by the sum $\sum_{i,j} x_{i,j} c_{ij}$. We want to minimize this cost.
The first thing to note is that if there is a cycle in this graph with positive edge weights \( x_{ij} \), then there is another distribution with weakly lower cost. Let the cycle be
\[
  w_{e_1} \rightarrow s_{e_1} \leftarrow w_{e_2} \rightarrow s_{e_2} \leftarrow \ldots
\]
If we lower each \( \rightarrow \) by \( \Delta \) and raise each \( \leftarrow \) by \( \Delta \), then each store receives the same net amount and each warehouse ships the same net amount, but cost changes accordingly. So we can zero out one of these edges without increasing overall cost. This allows us to just consider acyclic solutions.

Now assign numbers to each vertex \( w_i \) and \( s_j \), \( p(w_i) \) and \( p(s_j) \), so that in all cases \( p(s_j) - p(w_i) = c_{ij} \) along edges with positive weight. This is possible if there are no cycles (illustrate what can happen when there’s a cycle, prove that this is always doable by clipping leaf edges). Note that we now have
\[
  \text{cost} = \sum_{i,j \neq 0} x_{ij}c_{ij} = \sum_{i,j} x_{ij}(p(s_i) - p(w_j)) = \\
  \sum_{i=1}^{m} p(s_i) \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} p(w_j) \sum_{i=1}^{m} x_{ij} = \sum_{i=1}^{m} p(s_i)d_i - \sum_{j=1}^{n} p(w_i)s_i.
\]

To minimize the overall cost of shipping, we look for adjustments to the spanning tree we are dealing with to see if we can reduce the value of \( \sum_{i=1}^{m} p(s_i)d_i - \sum_{j=1}^{n} p(w_i)s_i \). Look at non-edges and any price adjustments that the addition of this edge necessitates. Add the edge that would imply the greatest reduction in cost, create cycle, then remove another cycle edge as above to lower the cost.