In a transportation problem, suppose the total supply $s$ produced exceeds the total demand $d$, $d < s$.

Solution: cut back on the supply produced.

Should each plant cut back equally? No, the ones with the biggest shipping costs should be cut the most.

We reduce this case to the previous $d = s$ case by adding a new “fictitious” warehouse $W_{n+1}$ for a new demand $d_{n+1}$.

Let $d_{n+1} = s - d$. With this additional demand we once again have: total demand = total supply.

The amount a plant ships to $W_{n+1} =$ the amount of supply which the plant should cut back.

Let the shipping cost $c_{i,n+1}$ from plant $P_i$ to $W_{n+1}$ be 0. Since nothing is really shipped, the cost is 0. We want to minimize the shipping costs to the real $W_j$. With $c_{i,n+1} = 0$ the total shipping cost = the real shipping cost.

Hungarian Algorithm

**GIVEN.** An $n \times n$ matrix.

**DEFINITION.** A set of 0’s is noncolinear iff no two 0’s are in the same row or in the same column.

**THEOREM.** The maximum number of noncolinear 0’s = the minimum number of horizontal and/or vertical lines needed to cross off all the zeros.

**COROLLARY.** Either there are $n$ noncolinear 0’s (every row and every col has one 0*) or all the 0’s can be crossed off with $< n$ lines.

**PROOF OF COROLLARY.** The Hungarian algorithm (which follows) always finds $n$ noncolinear 0’s (it marks them with a *) or finds $< n$ lines which cross off all 0’s.

- At each stage stars and primes may be added to or removed from uncrossed 0’s. Crossed-off 0’s are inactive.
- No row or column ever gets more than one 0*.
- If no further activity is possible for a 0*, its row or column is crossed off with a line. All 0’s in the line become crossed. Nothing more happens to crossed 0’s.
- A 0 is untouched if it is not crossed, starred or primed.
- Star-prime a 0 means place a star on the 0 and put primes on all other uncrossed 0’s in its row and column.
- A 0-0* path is a path whose corners are uncrossed 0’s such that no three elements are in the same column or in the same row. The 0’s are alternately starred and primed starting with an unstarring.

Whenever there are two or more choices, always pick the first choice. For rows first = highest, for columns first = leftmost.

### Hungarian Algorithm

**Given:** an $n \times n$ matrix.

**Output:**
1. $n$ noncolinear 0’s marked with a *.
2. $< n$ lines which cover all 0’s.

If a row (column) has no 0’s cross off all other rows (columns respectively). Stop.

**A:** While there is a row or column with a single uncrossed 0, star-prime the first such 0 and cross off either its row or its column, which ever crosses off the most uncrossed 0’s. Repeat while until done.

If all 0’s are crossed off, stop.

**B:** While there is an untouched 0: pick one in a row or column with the fewest untouched 0’s. Star-prime the first such 0. Repeat while.

If there are $n$ *0’s, stop.

Pick the first 0*. Pick a 0* from its row or column to start a 0*-0* path.

**C:** Extend this 0*-0* path as far as possible in both directions.
- If it begins and ends in 0*, replace each 0* along the path with a 0* and each 0* with 0*. Go to B.
- If the path begins or ends with 0*, cross off the row or column with this 0* and the path’s adjacent 0*.
- If the shortened path is nonempty, go to C; otherwise go to A.

- Run the algorithm on each of the following examples. Number the lines 1, 2, 3, ... in the order they are drawn.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$0* 1 0 0 0 1 1 0 0 0 1
1 0* 0 0 2 0 4
0 0 1 0 5 0 7
0 6 0 7

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For these simple problems, it’s easier to see an answer than to run the algorithm. Since just getting an answer is trivial for simple problems like these, no homework or exam credit will be given for answers not obtained by the algorithm.

In general, the nonzero values will be positive integers, e.g. 3, 9, ... , not just 1. ?? maybe simplify the algorithm to just the last steps.