Try to leave every 3rd row (mostly) empty

- Prelim 1 tonight 7:30 PM
  - NetID ends in 0-6: Olin Hall 155
  - NetID ends in 7-9: Olin Hall 165

```python
if rem(row, 3) == 0:
    % skip row
end
```
Previous Lecture:
- Introduction to 2-d array—matrix

Today, Lecture 14:
- Examples on computing with matrices
- Optional reading: contour plot (7.2, 7.3 in Insight)

Announcements:
- Prelim 1 tonight at 7:30pm
  - Olin Hall 155 & 165
- Discussion section in Upson 225 computer lab this week
Pattern for traversing a matrix $M$

\[
[nr, nc] = \text{size}(M)
\]

\[
\text{for } r = 1:nr
\]
\[
\quad \text{At row } r
\]
\[
\quad \text{for } c = 1:nc
\]
\[
\quad \quad \text{At column } c \text{ (in row } r)\n\]
\[
\quad \quad \%
\]
\[
\quad \quad \% \text{ Do something with } M(r,c) \ldots
\]
\[
\quad \end
\]
\[
\end
\]
Exercise: what’s different about this version?

function val = minInMatrix(M)
val = M(1,1);
[nr, nc] = size(M);
for c = 1:nc
    for r = 1:nr
        if M(r,c) < val
            val = M(r,c);
        end
    end
end

A: Nothing (identical to previous version)
B: Searches elements in a different order
C: Index-out-of-bounds if M is not square
D: Doesn’t correctly return minimum
Storing and using data in **tables**

A merchant has 3 suppliers that stock 5 products with these costs:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>36</td>
<td>22</td>
<td>15</td>
<td>62</td>
</tr>
<tr>
<td>12</td>
<td>35</td>
<td>20</td>
<td>12</td>
<td>66</td>
</tr>
<tr>
<td>13</td>
<td>37</td>
<td>21</td>
<td>16</td>
<td>59</td>
</tr>
</tbody>
</table>

What is the best way to fill a given purchase order?

Connections
between webpages:

```
0 0 1 0 1 0 0
1 0 0 1 1 1 0
0 1 0 1 1 1 1
1 0 1 1 0 1 0
0 0 1 1 0 1 1
0 0 1 0 1 0 1
0 1 1 0 1 1 0
```
A Cost/Inventory Problem

- A merchant has 3 supplier warehouses that stock 5 different products
- The cost of a product varies from warehouse to warehouse
- The inventory varies from warehouse to warehouse
Problems

A customer submits a purchase order that is to be shipped from a single warehouse.

1. How much would it cost a warehouse to fill the order?
2. Does a warehouse have enough inventory to fill the order?
3. Among the warehouses that can fill the order, who can do it most cheaply?
Available data

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>36</td>
<td>22</td>
<td>15</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>35</td>
<td>20</td>
<td>12</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>37</td>
<td>21</td>
<td>16</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>5</td>
<td>99</td>
<td>34</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**C(i,j)** is what it costs warehouse i to supply product j.

**Inv(i,j)** is the inventory in warehouse i of product j.

**PO(j)** is the number of product j’s that the client wants.
<table>
<thead>
<tr>
<th>C</th>
<th>10</th>
<th>36</th>
<th>22</th>
<th>15</th>
<th>62</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>35</td>
<td>20</td>
<td>12</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>37</td>
<td>21</td>
<td>16</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>PO</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
</tr>
</tbody>
</table>

Cost for warehouse 1:

\[1 \times 10 + 0 \times 36 + 12 \times 22 + 29 \times 15 + 5 \times 62\]
Cost for warehouse 1:

\[
\begin{align*}
\text{s} &= 0; \quad \% \text{Sum of cost} \\
\text{for j}=1:5 \\
\text{s} &= \text{s} + C(1,j) \times \text{PO}(j) \\
\text{end}
\end{align*}
\]
Cost for warehouse 2:

```matlab
s = 0;  % Sum of cost
for j=1:5
    s = s + C(2,j)*PO(j)
end
```
Cost for warehouse $i$:

$$s = 0; \quad %\text{Sum of cost}$$

for $j=1:5$

\[
s = s + C(i,j) \times PO(j)
\]

end
function theBill = iCost(i,C,PO)
% The cost when warehouse i fills the
% purchase order

nProd= length(PO);  % or size(c,2)
theBill= 0;
for j = 1:nProd
    theBill= theBill + C(i,j)*PO(j);
end
Finding the Cheapest

Both which warehouse and how cheap

```matlab
iBest = 0; minBill = inf;
for i = 1:nSuppliers
    iBill = iCost(i, C, PO);
    if iBill < minBill
        % Found an Improvement
        iBest = i; minBill = iBill;
    end
end
```
Aside: floating-point “bonus numbers”

- **inf**: Represents “infinity”
  - Both positive and negative versions
  - Larger (or smaller) than any other number
  - Generated on overflow or when dividing by zero

- **nan**: Not-a-number
  - Not equal to anything (even itself)
  - Not greater-than or less-than anything (even inf)
  - Generated from \(0/0, \inf*0\), ...
  - If involved in mathematical operation, result will be nan
What if a warehouse lacks the inventory to fill the purchase order?

Such a warehouse should be excluded from the find-the-cheapest computation.
Who Can Fill the Order?

<table>
<thead>
<tr>
<th>Inv</th>
<th>38</th>
<th>5</th>
<th>99</th>
<th>34</th>
<th>42</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
<td>Yes</td>
</tr>
<tr>
<td>PO</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Wanted: A True/False Function

iCanDo

DO is “true” if warehouse \( i \) can fill the order.
DO is “false” if warehouse \( i \) cannot fill the order.
Example: Check inventory of warehouse 2

<table>
<thead>
<tr>
<th>Inv</th>
<th>38</th>
<th>5</th>
<th>99</th>
<th>34</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
</tr>
</tbody>
</table>

| PO  | 1  | 0 | 12 | 29 | 5  |

Method 1: check the inventory for every product
**Initialization**

<table>
<thead>
<tr>
<th>Inv</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>5</td>
<td>99</td>
<td>34</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PO</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Still True…

\[ \text{DO} = \text{DO} && (\text{Inv}(2,1) \geq \text{PO}(1)) \]
Still True…

\[
\text{DO} = \text{DO} \land (\text{Inv}(2,2) \geq \text{PO}(2))
\]

<table>
<thead>
<tr>
<th></th>
<th>Inv</th>
<th></th>
<th></th>
<th></th>
<th>DO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>38</td>
<td>5</td>
<td>99</td>
<td>34</td>
<td>42</td>
</tr>
<tr>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
<td>1</td>
</tr>
</tbody>
</table>

\text{PO} = [1, 0, 12, 29, 5]
Still True…

\[
\text{DO} = \text{DO} \land (\text{Inv}(2,3) \geq \text{PO}(3))
\]
No Longer True…

\[
\begin{array}{cccccc}
38 & 5 & 99 & 34 & 42 \\
82 & 19 & 83 & 12 & 42 \\
51 & 29 & 21 & 56 & 87 \\
\end{array}
\]

\[
\begin{array}{cccccc}
1 & 0 & 12 & 29 & 5 \\
\end{array}
\]

\[
DO = DO && ( \text{Inv}(2,4) >= PO(4) )
\]
Stay False…

\[ DO = DO \land (\text{Inv}(2,5) \geq \text{PO}(5)) \]
function   DO = iCanDo(i,Inv,PO)
% DO is true if warehouse i can fill
% the purchase order. Otherwise, false

nProd= length(PO);
DO= 1;
for j = 1:nProd
    DO= DO && ( Inv(i,j) >= PO(j) );
end
function  DO = iCanDo(i,Inv,PO)
% DO is true if warehouse i can fill
% the purchase order. Otherwise, false
nProd= length(PO);
j= 1;
while j<=nProd && Inv(i,j)>=PO(j)
   j= j + 1;
end
DO= __________;

Method 2: stop as soon as you find one product for which there isn't enough inventory
function DO = iCanDo(i, Inv, PO)
% DO is true if warehouse i can fill % the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j <= nProd && Inv(i, j) >= PO(j)
    j = j + 1;
end
DO = __________;

DO should be true when...
A j < nProd
B j == nProd
C j > nProd
function DO = iCanDo(i, Inv, PO)
% DO is true if warehouse i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j <= nProd && Inv(i, j) >= PO(j)
    j = j + 1;
end
DO = (j > nProd);
iBest = 0; minBill = inf;
for i = 1:nSuppliers
    iBill = iCost(i,C,PO);
    if iBill < minBill
        % Found an Improvement
        iBest = i; minBill = iBill;
    end
end

Back To Finding the Cheapest

Don't bother with this unless there is sufficient inventory.
iBest = 0; minBill = inf;
for i = 1:nSuppliers
    if iCanDo(i,Inv,PO)
        iBill = iCost(i,C,PO);
        if iBill < minBill
            % Found an Improvement
            iBest = i; minBill = iBill;
        end
    end
end

See Cheapest.m for alternative implementation
### Finding the Cheapest

<table>
<thead>
<tr>
<th>C</th>
<th>10</th>
<th>36</th>
<th>22</th>
<th>15</th>
<th>62</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>35</td>
<td>20</td>
<td>12</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>37</td>
<td>21</td>
<td>16</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>PO</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
</tr>
</tbody>
</table>

- **1019**: Yes
- **930**: No
- **1040**: Yes

As computed by **iCost**

As computed by **iCanDo**
Matrix example: Random Web

- N web pages can be represented by an N-by-N Link Array $A$.
- $A(i,j)$ is 1 if there is a link on webpage $j$ to webpage $i$
N web pages can be represented by an N-by-N Link Array $A$.

$A(i,j)$ is 1 if there is a link on webpage $j$ to webpage $i$.

**Generate a random link array** and display the connectivity:

- There is no link from a page to itself.
- If $i \neq j$ then $A(i,j) = 1$ with probability $\frac{1}{1+|i-j|}$.

There is more likely to be a link if $i$ is close to $j$. 
function A = RandomLinks(n)
% A is n-by-n matrix of 1s and 0s
% representing n webpages

A = zeros(n,n); % initialize to 0s
for i = 1:n
  for j = 1:n
    % if A(i,j) not on diagonal,
    % assign 1 with some probability
  end
end
end
An event happens with probability $p$, $0 < p < 1$

% Flip a fair coin
r = rand();
if r < .5
    disp('heads')
else
    disp('tails')
end

% Unfair coin: shows heads twice as often as tails
r = rand();
if r < 2/3
    disp('heads')
else
    disp('tails')
end

% Event X happens with probability p
r = rand();
if r < p
    % Code for event X
end
function A = RandomLinks(n)
% A is n-by-n matrix of 1s and 0s
% representing n webpages

A = zeros(n,n);
for i = 1:n
    for j = 1:n
        r = rand();
        if i ~= j && r < 1/(1 + abs(i-j));
            A(i,j) = 1;
        end
    end
end
Random web: N=20

\[
\begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}

\text{M}(3,2) \quad \text{M}(2,3)
Represent the web pages graphically…

100 Web pages arranged in a circle.
Next display the links…. 
100 Web pages arranged in a circle. Bidirectional links are blue. Unidirectional link is black as it leaves page j, red when it arrives at page i.
for i = 1:n
    for j = 1:n
        if A(i,j) == 1 && A(j,i) == 1
            % Blue
        elseif A(i,j) == 1
            % Black
            j = mid
            mid = i
        end
    end
end

Somewhat inefficient: each blue line gets drawn twice.
See ShowRandomLinks.m
<table>
<thead>
<tr>
<th></th>
<th>A(1,3)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Transpose—like switching row and column indices*