CS381 Lecture Notes on
Largest Square Block of Ones

Finding a largest square block of 1s in a Boolean matrix

Let A be an $n \times n$ matrix of 0's and 1's. We seek an $O(n^2)$ time algorithm for finding the largest square block of A that contains 1's only. For example, the matrix shown below has a largest subblock of size 3 (the 3×3 square whose top-left corner is at the second row and third column).

```
0  1  1  0  1  0
0  0  1  1  1  0
1  1  1  1  1  1
1  1  1  1  1  0
1  0  0  1  1  1
1  0  0  0  1  1
```

Solution

Let M denote a matrix where $M[i, j]$ is the length of the side of the largest square block of 1's in A whose top-left corner is at position $[i, j]$ in A. Observe that $M[i, j] = 0$ if $A[i, j] = 0$, otherwise we have

$$M[i, j] = 1 + \min\{M[i + 1, j + 1], M[i + 1, j], M[i, j + 1]\}$$

(with the convention that out-of-bounds indices indicate a zero value). This implies that we can compute $M[i, j]$ in $O(1)$ time if we already know $M[i + 1, j + 1], M[i + 1, j], M[i, j + 1]$. Proceeding row by row, from the last row to the first and in right-to-left order within each row, we can therefore compute all of the $M[i, j]$s in $O(n^2)$ time. During this computation of the $M[i, j]$s we keep track of the largest $M[i, j]$ encountered so far (this will be the answer, after we finish computing the first row of M).

Note that the additional space for the above process (that is, the space used in addition to A itself) can be decreased from n^2 down to $O(n)$ because, when computing row i of M, we only need row $i + 1$ of M (so there is no need to store the other rows of M).