1.3 Array Search

Andre Monteiro

Term 3, 2025

(a) Assume a 3 x 3 matrix of form:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

wherein valid entries for k are highlighted in green, invalid in red, and our queried target in black.

- **Case 1:** $A[1][n] > k$.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

  k cannot appear in the last column or at $A[1][n]$ as subsequent elements must be strictly larger than $A[1][n]$, but may appear in the submatrix left of the last column,

  Since these positions are strictly less than elements below or to their right but may still equal k, they are potential candidates.

- **Case 2:** $A[1][n] < k$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

  k cannot appear in the first row or at $A[1][n]$ as the previous elements must be strictly smaller than $A[1][n]$, but may appear in the submatrix below the first row and left of the last column, or in the last column below the first row.

  Since these positions are strictly greater than elements above or to their left but may still equal k, they are potential candidates.

- **Case 3:** $A[1][n] = k$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

  As $A[1][n] = k$, the previous row elements must be smaller, and the column element larger. This implies only $A[1][n]$ and the subarray not intersecting with either column or row of $A[1][n]$, are valid targets for k.

  The element matches k, so no further search is technically needed.

Algorithm

We describe an algorithm to determine whether a target value k is present in a matrix A where each row and each column is sorted in increasing order.

1. Start at the top-right corner of the matrix

- Begin with the element in the first row and last column ($i = 1, j = n$).

  Justification: The top-right element is the largest in its row and the smallest in its column. This position allows us to make a clear decision at each step: whether to move left or down depending on how the element compares to k.

2. While $i \leq n$ and $j \geq 1$, compare $A[i][j]$ with k

- **If $A[i][j] = k$:** Report success and terminate. This is hte first case for early termination and avoids unncessary searching.

  Justification: Finding k satisfies the search condition; no further steps are necessary.

- **If $A[i][j] > k$:** Move one column to the left ($j \leftarrow j - 1$).

  Justification: All entries below $A[i][j]$ in the same column are greater than or equal to $A[i][j]$. Since $A[i][j] > k$, k cannot appear in this column below the current element. Moving left allows us to examine smaller elements in the current row.

- **If $A[i][j] < k$:** Move one row down ($i \leftarrow i + 1$).

  Justification: All entries to the left of $A[i][j]$ in the same row are less than or equal to $A[i][j]$. Since $A[i][j] < k$, k cannot appear in this row to the left. Moving down allows us to examine larger elements in the current column.

3. Termination If the search moves out of the bounds of the matrix without finding k, report that k is not present.

Justification: Each step eliminates at least one row or one column from consideration. Therefore, the algorithm cannot loop infinitely and will terminate. If k exists in the matrix, the algorithm will reach it, and if it does not exist, the algorithm will correctly report its absence.

Complexity Analysis

- **Time Complexity:** $O(n)$, since each step either moves down one row or left one column, and there are at most n rows and n columns.