# Arrays and Structures:

## Matrix Transpose

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

## Outline

1. Matrix Transpose

2. Fast Matrix Transpose

## Outline

1. Matrix Transpose
2. Fast Matrix Transpose

# Transposing a Matrix (1/4)

$M \in \mathbb{Z}^{6\times 6}$:

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

## Transposing a Matrix (1/4)

$M \in \mathbb{Z}^{6\times 6}$:

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

$M^\top \in \mathbb{Z}^{6\times 6}$:

$$\begin{bmatrix} 15 & 0 & 0 & 0 & 91 & 0 \\ 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 28 \\ 22 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -15 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

## Transposing a Matrix (2/4)

$M \in \mathbb{Z}^{6\times 6}$:

| | Row | Col | Value |
|---|---|---|---|
| A[0] | 6 | 6 | 8 |
| A[1] | $0$ | $0$ | $15$ |
| A[2] | $0$ | $3$ | $22$ |
| A[3] | $0$ | $5$ | $-15$ |
| A[4] | $1$ | $1$ | $11$ |
| A[5] | $1$ | $2$ | $3$ |
| A[6] | $2$ | $3$ | $-6$ |
| A[7] | $4$ | $0$ | $91$ |
| A[8] | $5$ | $2$ | $28$ |

# Transposing a Matrix (2/4)

$M \in \mathbb{Z}^{6\times 6}$:

| | Row | Col | Value |
|---|---|---|---|
| A[0] | 6 | 6 | 8 |
| A[1] | 0 | 0 | 15 |
| A[2] | 0 | 3 | 22 |
| A[3] | 0 | 5 | $-15$ |
| A[4] | 1 | 1 | 11 |
| A[5] | 1 | 2 | 3 |
| A[6] | 2 | 3 | $-6$ |
| A[7] | 4 | 0 | 91 |
| A[8] | 5 | 2 | 28 |

$M^{\top} \in \mathbb{Z}^{6\times 6}$:

| | Row | Col | Value |
|---|---|---|---|
| A[0] | 6 | 6 | 8 |
| A[1] | 0 | 0 | 15 |
| A[2] | 0 | 4 | 91 |
| A[3] | 1 | 1 | 11 |
| A[4] | 2 | 1 | 3 |
| A[5] | 2 | 5 | 28 |
| A[6] | 3 | 0 | 22 |
| A[7] | 3 | 2 | $-6$ |
| A[8] | 5 | 0 | $-15$ |

# Transposing a Matrix (3/4)

## Algorithm 1

- for each row $i$,
  - place element $\langle i, j, \text{value}\rangle$ in element $\langle j, i, \text{value}\rangle$

## Algorithm 2

- for each column $j$,
  - place element $\langle i, j, \text{value}\rangle$ in element $\langle j, i, \text{value}\rangle$

# Transposing a Matrix (3/4)

## Algorithm 1

- for each row $i$,
  - place element $\langle i, j, \text{value}\rangle$ in element $\langle j, i, \text{value}\rangle$

## Algorithm 2

- for each column $j$,
  - place element $\langle i, j, \text{value}\rangle$ in element $\langle j, i, \text{value}\rangle$

- What's the difficulty for Algorithm 1?

# Transposing a Matrix (4/4) $O$(columns × elements)

```c
void transpose(term a[], term b[]) { // b is set to the transpose of a
    int n, i, j, currentb;
    n = a[0].value; // total number of elements
    b[0].row = a[0].col; // rows in b = columns in a
    b[0].col = a[0].row; // columns in b = rows in a
    b[0].value = n;
    if (n > 0) { // dealing with a nonzero matrix
        currentb = 1;
        for (i=0; i<a[0].col; i++) // transpose by the columns in a
            for (j=1; j<=n; j++) // find elements from the current column
                if (a[j].col == i) { // element is in current column, add it to b
                    b[currentb].row = a[j].col;
                    b[currentb].col = a[j].row;
                    b[currentb].value = a[j].value;
                    currentb++;
                }
    }
}
```

# Time Complexity of the Transpose Algorithm

- For matrices represented as 2-D arrays, the time complexity for computing a matrix transpose is $O(\text{columns} \times \text{rows})$.

# Time Complexity of the Transpose Algorithm

- For matrices represented as 2-D arrays, the time complexity for computing a matrix transpose is $O(\text{columns} \times \text{rows})$.
- The complexity of the previous transpose algorithm:
  - $O(\text{columns} \times \text{elements})$.
  - $O(\text{columns} \times \textbf{columns} \times \text{rows})$ if elements $\approx$ columns × rows.

# Time Complexity of the Transpose Algorithm

- For matrices represented as 2-D arrays, the time complexity for computing a matrix transpose is $O(\text{columns} \times \text{rows})$.
- The complexity of the previous transpose algorithm:
  - $O(\text{columns} \times \text{elements})$.
  - $O(\text{columns} \times \textbf{columns} \times \text{rows})$ if elements $\approx$ columns $\times$ rows.
- **Issue:** Scan the array for "#columns" times.

# Alternative Solution

- Determine the starting positions of each row in the transpose matrix.

# Alternative Solution

- Determine the starting positions of each row in the transpose matrix.
- Determine the number of elements in each column of the original matrix.

# Alternative Solution

- Determine the starting positions of each row in the transpose matrix.
- Determine the number of elements in each column of the original matrix.

| | Row | Col | Value |
|---|---|---|---|
| A[0] | 6 | 6 | 8 |
| A[1] | 0 | 0 | 15 |
| A[2] | 0 | 3 | 22 |
| A[3] | 0 | 5 | −15 |
| A[4] | 1 | 1 | 11 |
| A[5] | 1 | 2 | 3 |
| A[6] | 2 | 3 | −6 |
| A[7] | 4 | 0 | 91 |
| A[8] | 5 | 2 | 28 |

| | [0] | [1] | [2] | [3] | [4] | [5] |
|---|---|---|---|---|---|---|
| row_items | 2 | 1 | 2 | 2 | 0 | 1 |
| starting_pos | 1 | 3 | 4 | 6 | 8 | 8 |

# Outline

1. Matrix Transpose

2. Fast Matrix Transpose

# Fast Transposing a Matrix ($O(\text{columns} + \text{elements})$)

```
void fast_transpose(term a[], term b[]) { // b is set to the transpose of a
    int row_terms[MAX_COL], starting_pos[MAX_COL];
    int i, j, num_cols = a[0].col, num_terms = a[0].value;
    b[0].row = num_cols; b[0].col = a[0].row; b[0].value = num_terms;
    if (num_terms > 0) { // nonzero matrix
        for (i = 0; i < num_cols; i++)
            row_terms[i] = 0; // initialization for the counting
        for (i = 1; i <= num_terms; i++)
            row_terms[a[i].col]++; // counting the row items
        starting_pos[0] = 1;
        for (i = 1; i < num_cols; i++)
            starting_pos[i] = starting_pos[i-1] + row_items[i-1];
        for (i = 1; i <= num_terms; i++) {
            j = starting_pos[a[i].col]++;
            b[j].row = a[i].col;
            b[j].col = a[i].row;
            b[j].value = a[i].value;
        }
    }
}
```

# Time Complexity of the Fast Transpose Algorithm

- $O(\text{columns} + \text{elements})$.
- $O(\text{columns} + \textbf{columns} \times \text{rows})$ if elements $\approx$ columns × rows.
- Additional `row_terms` and `starting_pos` arrays are required.

# Discussions