

# Arrays and Structures: Matrix Transpose & Multiplication

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,  
National Taiwan Ocean University

Fall 2024

# Outline

## ① Matrix Transpose

- Fast Matrix Transpose

## ② Matrix Multiplication

# Outline

## 1 Matrix Transpose

- Fast Matrix Transpose

## 2 Matrix Multiplication

# 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) $O(\text{columns} \times \text{elements})$

```
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:
  - $O(\text{columns} \times \text{elements})$ .
  - $O(\text{columns} \times \text{columns} \times \text{rows})$  if elements  $\approx \text{columns} \times \text{rows}$ .

# 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:
  - $O(\text{columns} \times \text{elements})$ .
  - $O(\text{columns} \times \text{columns} \times \text{rows})$  if elements  $\approx \text{columns} \times \text{rows}$ .
- 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.

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

	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

- Scan over  $a[0].value$  times to count `row_terms`.

# Outline

## 1 Matrix Transpose

- Fast Matrix Transpose

## 2 Matrix Multiplication

# 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]++;
        starting_pos[0] = 1;
        for (i = 1; i < num_cols; i++) // calculate the starting positions
            starting_pos[i] = starting_pos[i-1] + row_terms[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

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

# Outline

## ① Matrix Transpose

- Fast Matrix Transpose

## ② Matrix Multiplication

# Matrix Multiplication

## Matrix Multiplication

Given an  $m \times n$  matrix  $A$  and an  $n \times p$  matrix  $B$ , the product  $D = A \times B$  is of shape  $m \times p$ , such that its  $(i, j)$ -th element is

$$d_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

for  $0 \leq i \leq m$  and  $0 \leq j \leq p$ .

- For example,

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 3 & -1 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ -2 & -1 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 0 & -5 \\ -6 & -7 \end{bmatrix}$$

# Matrix Multiplication

## Matrix Multiplication

Given an  $m \times n$  matrix  $A$  and an  $n \times p$  matrix  $B$ , the product  $D = A \times B$  is of shape  $m \times p$ , such that its  $(i, j)$ -th element is

$$d_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

for  $0 \leq i \leq m$  and  $0 \leq j \leq p$ .

- For example,

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 3 & -1 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ -2 & -1 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 0 & -5 \\ -6 & -7 \end{bmatrix}$$

# Matrix Multiplication

## Matrix Multiplication

Given an  $m \times n$  matrix  $A$  and an  $n \times p$  matrix  $B$ , the product  $D = A \times B$  is of shape  $m \times p$ , such that its  $(i, j)$ -th element is

$$d_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

for  $0 \leq i \leq m$  and  $0 \leq j \leq p$ .

- For example,

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 3 & -1 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ -2 & -1 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 0 & -5 \\ -6 & -7 \end{bmatrix}$$

# Matrix Multiplication

## Matrix Multiplication

Given an  $m \times n$  matrix  $A$  and an  $n \times p$  matrix  $B$ , the product  $D = A \times B$  is of shape  $m \times p$ , such that its  $(i, j)$ -th element is

$$d_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

for  $0 \leq i \leq m$  and  $0 \leq j \leq p$ .

- For example,

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 3 & -1 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ -2 & -1 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 0 & -5 \\ -6 & -7 \end{bmatrix}$$

# Matrix Multiplication

## Matrix Multiplication

Given an  $m \times n$  matrix  $A$  and an  $n \times p$  matrix  $B$ , the product  $D = A \times B$  is of shape  $m \times p$ , such that its  $(i, j)$ -th element is

$$d_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

for  $0 \leq i \leq m$  and  $0 \leq j \leq p$ .

- For example,

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 3 & -1 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ -2 & -1 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 0 & -5 \\ -6 & -7 \end{bmatrix}$$

# Using Matrix Transpose

$$A = \begin{bmatrix} -27 & 6 \\ 82 & 0 \\ 109 & -64 \end{bmatrix} :$$

$$B = \begin{bmatrix} 21 & 27 & 0 \\ 5 & 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
A[0]	3	2	5
A[1]	0	0	-27
A[2]	0	1	6
A[3]	1	0	82
A[4]	2	0	109
A[5]	2	1	-64
A[6]	3		

	Row	Col	Value
B[0]	2	3	4
B[1]	0	0	21
B[2]	0	1	27
B[3]	1	0	5
B[4]	1	2	-71

# Using Matrix Transpose

$$A = \begin{bmatrix} -27 & 6 \\ 82 & 0 \\ 109 & -64 \end{bmatrix} :$$

	Row	Col	Value
A[0]	3	2	5
A[1]	0	0	-27
A[2]	0	1	6
A[3]	1	0	82
A[4]	2	0	109
A[5]	2	1	-64
A[6]	3		

$$B = \begin{bmatrix} 21 & 27 & 0 \\ 5 & 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
B[0]	2	3	4
B[1]	0	0	21
B[2]	0	1	27
B[3]	1	0	5
B[4]	1	2	-71

$$B^T = \begin{bmatrix} 21 & 5 \\ 27 & 0 \\ 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
$B^T[0]$	3	2	4
$B^T[1]$	0	0	21
$B^T[2]$	0	1	5
$B^T[3]$	1	0	27
$B^T[4]$	2	1	-71
$B^T[5]$	3	-1	

boundary condition

# Sparse Matrix Multiplication

$$A = \begin{bmatrix} -27 & 6 \\ 82 & 0 \\ 109 & -64 \end{bmatrix} :$$

	Row	Col	Value
A[0]	3	2	5
A[1]	0	0	-27
A[2]	0	1	6
A[3]	1	0	82
A[4]	2	0	109
A[5]	2	1	-64
A[6]	3		

$$B = \begin{bmatrix} 21 & 27 & 0 \\ 5 & 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
B[0]	2	3	4
B[1]	0	0	21
B[2]	0	1	27
B[3]	1	0	5
B[4]	1	2	-71

$$B^\top = \begin{bmatrix} 21 & 5 \\ 27 & 0 \\ 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
$B^\top[0]$	3	2	4
$B^\top[1]$	0	0	21
$B^\top[2]$	0	1	5
$B^\top[3]$	1	0	27
$B^\top[4]$	2	1	-71
$B^\top[5]$	3	-1	

# Sparse Matrix Multiplication

$$A = \begin{bmatrix} -27 & 6 \\ 82 & 0 \\ 109 & -64 \end{bmatrix} :$$

	Row	Col	Value
A[0]	3	2	5
A[1]	0	0	-27
A[2]	0	1	6
A[3]	1	0	82
A[4]	2	0	109
A[5]	2	1	-64
A[6]	3		

$$B = \begin{bmatrix} 21 & 27 & 0 \\ 5 & 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
B[0]	2	3	4
B[1]	0	0	21
B[2]	0	1	27
B[3]	1	0	5
B[4]	1	2	-71

$$B^T = \begin{bmatrix} 21 & 5 \\ 27 & 0 \\ 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
$B^T[0]$	3	2	4
$B^T[1]$	0	0	21
$B^T[2]$	0	1	5
$B^T[3]$	1	0	27
$B^T[4]$	2	1	-71
$B^T[5]$	3	-1	

- Procedure: Fix a row of  $A$  and find all elements in column  $j$  of  $B$ , for  $j = 0, 1, \dots, p - 1$ .

# Sparse Matrix Multiplication

$$A = \begin{bmatrix} -27 & 6 \\ 82 & 0 \\ 109 & -64 \end{bmatrix} :$$

	Row	Col	Value
A[0]	3	2	5
A[1]	0	0	-27
A[2]	0	1	6
A[3]	1	0	82
A[4]	2	0	109
A[5]	2	1	-64
A[6]	3		

$$B = \begin{bmatrix} 21 & 27 & 0 \\ 5 & 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
B[0]	2	3	4
B[1]	0	0	21
B[2]	0	1	27
B[3]	1	0	5
B[4]	1	2	-71

$$B^T = \begin{bmatrix} 21 & 5 \\ 27 & 0 \\ 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
$B^T[0]$	3	2	4
$B^T[1]$	0	0	21
$B^T[2]$	0	1	5
$B^T[3]$	1	0	27
$B^T[4]$	2	1	-71
$B^T[5]$	3	-1	

- Procedure: Fix a row of  $A$  and find all elements in column  $j$  of  $B$ , for  $j = 0, 1, \dots, p - 1$ .
- Compute  $B^T$  (column elements are placed consecutively).

# Sparse Matrix Multiplication

$$A = \begin{bmatrix} -27 & 6 \\ 82 & 0 \\ 109 & -64 \end{bmatrix} :$$

$$B = \begin{bmatrix} 21 & 27 & 0 \\ 5 & 0 & -71 \end{bmatrix} :$$

$$B^T = \begin{bmatrix} 21 & 5 \\ 27 & 0 \\ 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
A[0]	3	2	5
A[1]	0	0	-27
A[2]	0	1	6
A[3]	1	0	82
A[4]	2	0	109
A[5]	2	1	-64
A[6]	3		

	Row	Col	Value
B[0]	2	3	4
B[1]	0	0	21
B[2]	0	1	27
B[3]	1	0	5
B[4]	1	2	-71

	Row	Col	Value
$B^T[0]$	3	2	4
$B^T[1]$	0	0	21
$B^T[2]$	0	1	5
$B^T[3]$	1	0	27
$B^T[4]$	2	1	-71
$B^T[5]$	3	-1	

- Procedure: Fix a row of  $A$  and find all elements in column  $j$  of  $B$ , for  $j = 0, 1, \dots, p - 1$ .
- Compute  $B^T$  (column elements are placed consecutively).

# Sparse Matrix Multiplication

$$A = \begin{bmatrix} -27 & 6 \\ 82 & 0 \\ 109 & -64 \end{bmatrix} :$$

$$B = \begin{bmatrix} 21 & 27 & 0 \\ 5 & 0 & -71 \end{bmatrix} :$$

$$B^T = \begin{bmatrix} 21 & 5 \\ 27 & 0 \\ 0 & -71 \end{bmatrix} :$$

	Row	Col	Value
A[0]	3	2	5
A[1]	0	0	-27
A[2]	0	1	6
A[3]	1	0	82
A[4]	2	0	109
A[5]	2	1	-64
A[6]	3		

	Row	Col	Value
B[0]	2	3	4
B[1]	0	0	21
B[2]	0	1	27
B[3]	1	0	5
B[4]	1	2	-71

	Row	Col	Value
$B^T[0]$	3	2	4
$B^T[1]$	0	0	21
$B^T[2]$	0	1	5
$B^T[3]$	1	0	27
$B^T[4]$	2	1	-71
$B^T[5]$	3	-1	

- Procedure: Fix a row of  $A$  and find all elements in column  $j$  of  $B$ , for  $j = 0, 1, \dots, p - 1$ .
- Compute  $B^T$  (column elements are placed consecutively).

# Rough Code HERE (Exercise)

```
void mat_mult(term a[], term b[], term d[]) { //multiply two sparse matrices
    int i, j, column, totalA = a[0].value, totalB = b[0].value, totalD = 0;
    int rowsA = a[0].row, colsA = a[0].col, colsB = b[0].col;
    int row_begin = 1, row = a[1].row, sum = 0;
    term newB[MAX_TERMS][3];
    if (colsA != b[0].row) {
        fprintf(stderr, "incompatible matrices\n");
        exit(0);
    }
    fast_transpose(b, newB);
    // set boundary conditions
    a[totalA+1].row = rowsA;
    newB[totalB+1].row = colsB;
    newB[totalB+1].col = 0;
```



## Matrix Multiplication

```
for (i=1; i<=totalA; ) {
    column = newB[i].row;
    for (j=1; j<=totalB+1; ) {
        // multiply row of a by column of b
        if (a[i].row != row) {
            storeSum(d, &totalD, row, column, &sum);
            i = row_begin;
            for (; newB[j].row == column; j++)
                ;
            column = newB[j].row;
        } else if (newB[j].row != column) {
            storeSum(d, &totalD, row, column, &sum);
            i = row_begin;
            column = newB[j].row;
        } else switch (COMPARE(a[i].col, newB[j].col)) {
            case -1: // go to next term of a
                i++; break;
            case 0: // add terms, go to next term in a and b
                sum += (a[i++].value * newB[j++].value); break;
            case 1: // advance to next term in b
                j++;
        } // end of for j <= totalB+1
        for (; a[i].row == row; i++)
            ;
        row_begin = i; row = a[i].row;
    } // end of for i<= totalA
    d[0].row = rowsA;
    d[0].col = colsB;
    d[0].value = totalD;
}
```

```
void storeSum(term d[], int *totalD, int row, int column, int *sum) {
    /* if *sum != 0, then it along with its row and column
       position is stored as the *totalD+1 entry in d */
    if (*sum) {
        if (*totalD < MAX_TERMS) {
            d[***totalD].row = row;
            d[*totalD].col = column;
            d[*totalD].value = *sum;
            *sum = 0;
        } else {
            fprintf(stderr, "Numbers of terms in product exceeds %d\n",
                    MAX_TERMS);
            exit(1);
        }
    }
}
```



# Asymptotic Time Complexity

- $\text{cols}B$ : the number of columns in matrix  $B$   
 $\text{rows}A$ : the number of rows in matrix  $A$   
 $\text{termsRow}_i$ : the number of terms in row  $i$  of  $A$   
 $\text{total}B$ : the number of elements in matrix  $B$ .
- Thus,

# Asymptotic Time Complexity

- $\text{cols}B$ : the number of columns in matrix  $B$   
 $\text{rows}A$ : the number of rows in matrix  $A$   
 $\text{termsRow}_i$ : the number of terms in row  $i$  of  $A$   
 $\text{total}B$ : the number of elements in matrix  $B$ .
- Thus,

$$\begin{aligned}& \text{cols}B \times \text{termsRow}_0 + \text{total}B \\&+ \text{cols}B \times \text{termsRow}_1 + \text{total}B \\&\quad \vdots \\&+ \text{cols}B \times \text{termsRow}_p + \text{total}B \\&= \text{cols}B \times \sum_{i=1}^p \text{termsRow}_i + \text{rows}A \times \text{total}B \\&= \text{cols}B \times \text{total}A + \text{rows}A \times \text{total}B.\end{aligned}$$

# Comparing with the classic matrix multiplication...

```
for (i=0; i<rowsA; i++) {  
    for (j=0; j<colsB; j++) {  
        fum = 0;  
        for (k=0; k<colsA; k++) {  
            sum += (a[i][k]*b[k][j]);  
        }  
        d[i][j] = sum;  
    }  
}
```

- The time complexity:  $O(\text{rows}A \times \text{cols}B \times \text{cols}A)$ .
- Sparse matrix multiplication:  $O(\text{cols}B \times \text{total}A + \text{rows}A \times \text{total}B)$ .
  - Optimal when  $\text{total}A < \text{rows}A \times \text{cols}A$  and  $\text{total}B < \text{cols}A \times \text{cols}B$ .
  - Worse when  $\text{total}A > \text{rows}A \times \text{cols}A$  or  $\text{total}B > \text{cols}A \times \text{cols}B$ .



