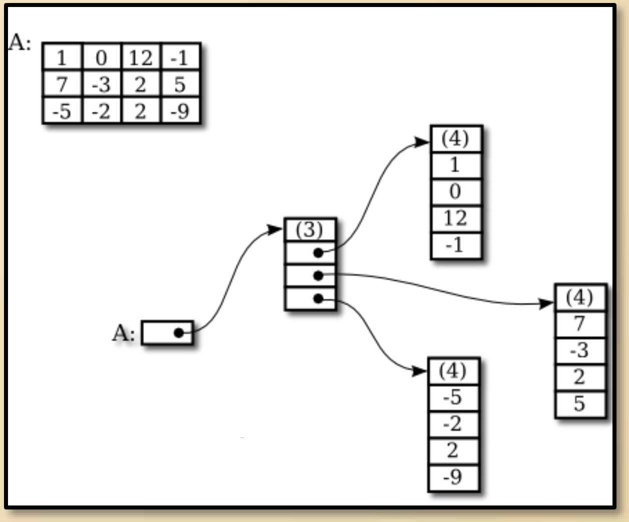
Introduction to Computer Science
Reichman University

Lecture 5-1

Two-Dimensional Arrays



A:

1	0	12	-1
7	-3	2	5
-5	-2	2	-9

Two dimensional arrays

movie rating: (1 to 5)

movie id

		0	1	2	3	4
	0	4	5	4	5	4
reviewer id	1	5	3	4	4	4
	2	3	4	3	5	3

Two dimensional arrays

movie rating: (1 to 5)

movie id

		0	1	2	3	4
	0	4	5	4	5	4
reviewer id	1	5	3	4	4	4
	2	3	4	3	5	3

sales data (units)

quarter

		1	2	3	4
	1	10	12	20	12
	2	12	14	25	18
model	3	14	16	22	17
	4	9	12	35	20
	5	11	15	30	22

image (black / white pixels)

column

		0	1	2	3	4	5	6	7	...
	0	0	0	0	0	0	0	0	0	0
	1	0	0	1	1	1	1	0	0	0
row	2	0	0	0	1	1	1	1	0	0
	3	0	0	0	0	1	1	1	1	0
	4	0	0	0	0	0	0	0	0	0
	...	0	0	0	0	0	0	0	0	0

image (greyscale pixels)

column

		0	1	2	3	4	5	6	7	...
	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
	2	0	255	255	255	255	255	0	0	0
row	3	0	112	17	17	83	83	0	0	0
	4	0	76	76	63	63	63	0	0	0
	...	0	0	0	0	0	0	0	0	0

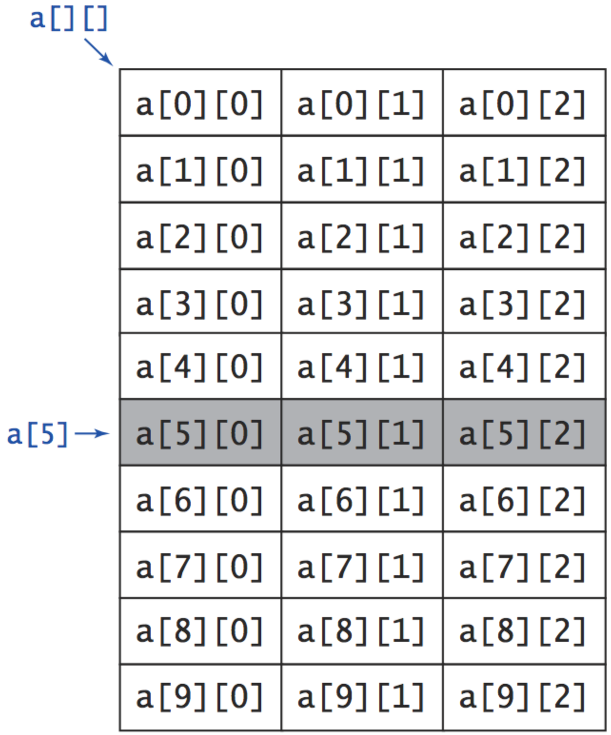
Two dimensional arrays

Technically speaking

A 2D array is an array of 1D arrays

"Row" `a[i]` is a 1D array whose elements are:

`a[i][0], a[i][1], a[i][2]`



a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]
a[3][0]	a[3][1]	a[3][2]
a[4][0]	a[4][1]	a[4][2]
a[5][0]	a[5][1]	a[5][2]
a[6][0]	a[6][1]	a[6][2]
a[7][0]	a[7][1]	a[7][2]
a[8][0]	a[8][1]	a[8][2]
a[9][0]	a[9][1]	a[9][2]

A 10-by-3 array

Conventions

- Row and column indexes start at 0
- The (row,col) element is accessed by `a[row][col]`
- Number of rows: `a.length`
- Number of elements in row i: `a[i].length`

Example

```
// Computes the sums of the rows in a table.
// Stores the sums in the rightmost column.
public class SumOfRow {
```

```
% java SumOfRow
 6 10  6 22
 7 11 17 35
 3 15  1 19
14  6 13 33
13  6  0 19
```

sum of each row

Example

```java
// Computes the sums of the rows in a table.
// Stores the sums in the rightmost column.
public class SumOfRow {
    public static void main(String[] args) {
        // Defines a 5 by 4 2D array, and puts random values in the
        // range 0...19 in the leftmost 3 columns, for testing
        int[][] arr = new int[5][4];

        for (int i = 0; i < 5; i++)  {
            for (int j = 0; j < 3; j++) {
                arr[i][j] = (int) (20 * Math.random());
            }
        }

        // Stores the sum of each row at the rightmost column
        for (int i = 0; i < 5; i++)  {
            int sum = 0;
            for (int j = 0; j < 3; j++) {
                sum = sum + arr[i][j];
            }
            arr[i][3] = sum;
        }
```

```
% java SumOfRow
 6 10  6 22
 7 11 17 35
 3 15  1 19
14  6 13 33
13  6  0 19
```

sum of each row

Example

```
// Computes the sums of the rows in a table.
// Stores the sums in the rightmost column.
public class SumOfRow {
    public static void main(String[] args) {
        // Defines a 5 by 4 2D array, and puts random values in the
        // range 0...19 in the leftmost 3 columns, for testing
        int[][] arr = new int[5][4];

        for (int i = 0; i < 5; i++)  {
            for (int j = 0; j < 3; j++) {
                arr[i][j] = (int) (20 * Math.random());
            }
        }

        // Stores the sum of each row at the rightmost column
        for (int i = 0; i < 5; i++)  {
            int sum = 0;
            for (int j = 0; j < 3; j++) {
                sum = sum + arr[i][j];
            }
            arr[i][3] = sum;
        }

        // Prints the array
        for (int i = 0; i < 5; i++)  {
            for (int j = 0; j < 4; j++) {
                System.out.printf("%4s", arr[i][j]);
            }
            System.out.println();
        }
    }
}
```

Prints the value using 4 positions, right justified.

```
% java SumOfRow
  6  10   6  22
  7  11  17  35
  3  15   1  19
 14   6  13  33
 13   6   0  19
```

sum of each row

Plan

- ✓ 2D arrays: Basic concepts
- ➡ Example: Matrix operations
- Internal view of 2D arrays
- Aside topic: Reading data from a file
- PageRank algorithm (example of 2D array processing)

Matrix operations

Addition

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

Precondition: The two matrices must have the same dimensions.

```java
// Computes the matrix addition sum = m1 + m2
// Assumes that m1, m2 have the same dimensions
int N = m1.length;      // number of rows
int M = m1[0].length;   // number of columns
int[][] sum = new int[N][M];
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        sum[i][j] = m1[i][j] + m2[i][j];
    }
}
```

Matrix operations

Multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$

Precondition: The two matrixes must have compatible dimensions:
the number of columns in the left matrix must equal the number
of rows in the right matrix.

Matrix operations

```
/** Features matrix operations. */
public class MatrixOps {
   // Illustrates using the class functions
   public static void main(String args[]) {
      int[][] a = { { 7, 2, 1 },
                    { 3, 6, 1 },
                    { 5, 1, 4 } };
      int[][] b = { { 8, 3, 5 },
                    { 1, 4, 1 },
                    { 1, 3, 4 } };
      println(a);
      println(b);
      println(add(a, b));
      println(mult(a, b));
      ...
```

```
/** Features matrix operations */
public class MatrixOps {
  /** Returns the matrix addition m1 + m2.
      Assumes that they have the same dimensions. */
   public static int[][] add(int[][] m1, int[][] m2)

   /** Returns the matrix multiplication m1 * m2.
      Assumes that they have compatible dimensions. */
   public static int[][] mult(int[][] m1, int[][] m2)

   /** Prints the given matrix. */
   public static void println(int[][] m)
}
```

MatrixOps API

```
% java MatrixOps
   7   2   1
   3   6   1      a
   5   1   4

   8   3   5
   1   4   1      b
   1   3   4

  15   5   6
   4  10   2      a + b
   6   4   8

  59  32  41
  31  36  25      a × b
  45  31  42
```

Matrix operations

```java
/**  Features matrix operations.  */
public class MatrixOps {
   // Illustrates using the class functions
   public static void main(String args[]) {
      int[][] a = { { 7, 2, 1 },
                    { 3, 6, 1 },
                    { 5, 1, 4 } };
      int[][] b = { { 8, 3, 5 },
                    { 1, 4, 1 },
                    { 1, 3, 4 } };
      println(a);
      println(b);
      println(add(a, b));
      println(mult(a, b));
      ...
      // Creates and computes c = a × (a + b)
      int[][] c = mult(a, add(a, b));

      // Computes c = (a + c) × b + c
      c = add((mult(add(a, c), b), c);
      ...
   }
   /**  Returns the addition of the two given matrices */
   public static int[][] add(...) {...}

   /**  Returns the product of the two given matrices */
   public static int[][] mult(...) {...}

   /**  Prints the given matrix.  */
   private static void println(int[][] m) {...}
}
```

```java
/** Features matrix operations */
public class MatrixOps {
   /** Returns the matrix addition m1 + m2.
       Assumes that they have the same dimensions. */
   public static int[][] add(int[][] m1, int[][] m2)

   /** Returns the matrix multiplication m1 * m2.
       Assumes that they have compatible dimensions. */
   public static int[][] mult(int[][] m1, int[][] m2)

   /** Prints the given matrix. */
   public static void println(int[][] m)
}
```

MatrixOps API

```
% java MatrixOps
   7   2   1
   3   6   1        a
   5   1   4

   8   3   5
   1   4   1        b
   1   3   4

  15   5   6
   4  10   2        a + b
   6   4   8

  59  32  41
  31  36  25        a × b
  45  31  42
```

Matrix operations

```
/**  Features matrix operations.  */
public class MatrixOps {
   // Illustrates using the class functions
   public static void main(String args[]) {
      int[][] a = { { 7, 2, 1 },
                    { 3, 6, 1 },
                    { 5, 1, 4 } };
      int[][] b = { { 8, 3, 5 },
                    { 1, 4, 1 },
                    { 1, 3, 4 } };
      println(a);
      println(b);
      println(add(a, b));
      println(mult(a, b));
      ...
      // Creates and computes c = a × (a + b)
      int[][] c = mult(a, add(a, b));

      // Computes c = (a + c) × b + c
      c = add((mult(add(a, c), b), c);
      ...
   }
   /**  Returns the addition of the two given matrices */
   public static int[][] add(...) {...}

   /**  Returns the product of the two given matrices */
   public static int[][] mult(...) {...}

   /**  Prints the given matrix.  */
   private static void println(int[][] m) {...}
}
```

```
/** Features matrix operations */
public class MatrixOps {
   /** Returns the matrix addition m1 + m2.
       Assumes that they have the same dimensions. */
   public static int[][] add(int[][] m1, int[][] m2)

   /** Returns the matrix multiplication m1 * m2.
       Assumes that they have compatible dimensions. */
   public static int[][] mult(int[][] m1, int[][] m2)

   /** Prints the given matrix. */
   public static void println(int[][] m)
}
```

MatrixOps API

```
% java MatrixOps
   7   2   1

  15   5   6
   4  10   2       a + b
   6   4   8

  59  32  41
  31  36  25       a × b
  45  31  42
```

Notice how add, mult, println are used as black box abstractions;

We will now open up the black boxes.

Matrix operations

Implementation

```
/** Features matrix operations. */
public class MatrixOps {
   public static void main(String args[]) {
      // Performs various tests (see previous slides)
   }
   /** Returns a matrix which is the addition of the two given matrices. Assumes they have the same dimensions. */
   public static int[][] add(int[][] m1, int[][] m2) {
      int N = m1.length;     // number of rows
      int M = m1[0].length; // number of columns
      int[][] sum = new int[N][M];
      for (int i = 0; i < N; i++) {
         for (int j = 0; j < M; j++) {
            sum[i][j] = m1[i][j] + m2[i][j];
         }
      }
      return sum;
   }
   /** Returns a matrix which is the product of the two given matrices. Assumes that they have compatible dimensions. */
   public static int[][] mult(int[][] m1, int[][] m2) {
      //// Replace the following statement with your code
      retrun null;
   }

   /** Prints the given matrix, and moves the cursor to the next line. */
   public static void println(int[][] m) {
      //// Similar to the sumOfRow example
   }
}
```

Plan

- ✓ 2D arrays: Basic concepts
- ✓ Example: Matrix operations
- ➡ Internal view of 2D arrays
- Aside topic: Reading data from a file
- PageRank algorithm (example of 2D array processing)

2D arrays: Abstraction and implementation

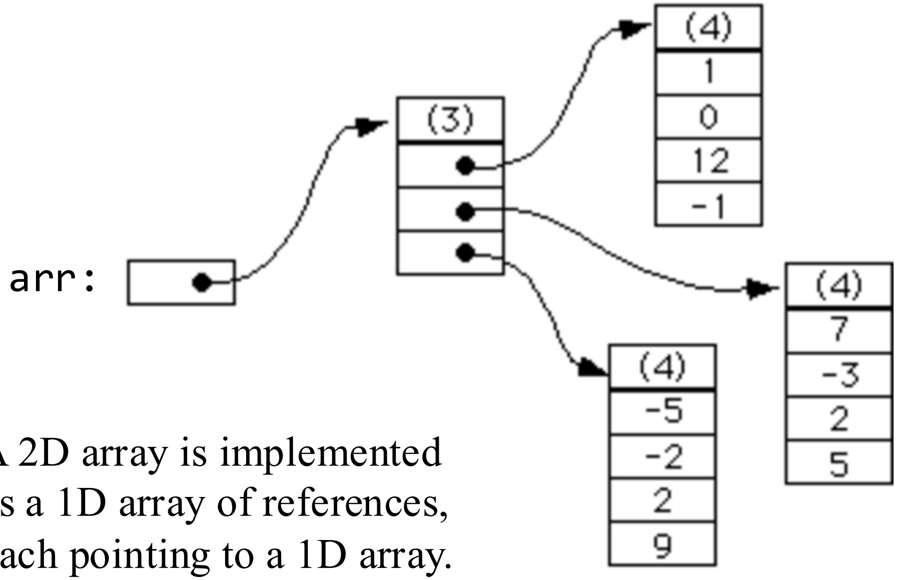
```
...
int[][] arr = { { 1,  0, 12, -1 },
                { 7, -3,  2,  5 },
                {-5, -2,  2,  9 }, };
...
```

Abstract view

arr:

1	0	12	-1
7	-3	2	5
-5	-2	2	9

Physical view



A 2D array is implemented as a 1D array of references, each pointing to a 1D array.