Data Structures [R1UC308B]

Module-II: Array

Dr. Subhash Chandra Gupta

School of Computer Science and Engineering
Plat No 2, Sector 17A, Yamuna Expressway
Greater Noida, Uttar Pradesh - 203201

July 31, 2025

Contents

Arrays

Here are the main properties of arrays in Java:

- Arrays are objects.
- Arrays are created dynamically (at run time).
- Any method of the Object class may be invoked on an array.
- The variables are called the components or elements of the array.
- If the component type is T, then the array itself has type T[].
- An element's type may be either primitive or reference.
- The length of an array is its number of components.
- An array's length is set when the array is created, and it cannot be changed.
- Array index values must be integers in the range 0...length –1.

- Variables of type short, byte, or char can be used as indexes.

Here are some valid array definitions:

- float x[ ] = new float[100];
- String[ ] args; args = new String[10];
- boolean[ ] isPrime = new boolean[1000];
- int fib[ ] = {0, 1, 1, 2, 3, 5, 8, 13};
- short[ ][ ][ ] b = new short[4][10][5];
- double a[ ][ ] = {{1.1, 2.2}, {3.3, 4.4}, *null*, {5.5, 6.6}, *null*};
  a[4] = new double[66];
  a[4][65] = 3.14;

Single and Multidimensional Arrays

- int a[ ];
- int a[ ][ ];
- int a[ ][ ][ ];

Representation of Arrays

- Row Major Order: Row major ordering assigns successive elements, moving across the rows and then down the next row, to successive memory locations. In simple language, the elements of an array are stored in a Row-Wise fashion.
- Column Major Order: If elements of an array are stored in a column-major fashion means moving across the column and then to the next column then it's in column-major order.

Derivation of Index Formula

- 1-D Array: Address of $A[Index] = B + W * (Index - LB)$
  where:
  Index = The index of the element whose address is to be found (not the value of the element).
  B = Base address of the array.
  W = Storage size of one element in bytes.
  LB = Lower bound of the index (if not specified, assume zero).

- 2-D Array:
  Row Major Order:
  Address of $A[I][J] = B + W * (M * (I–LR) + (J–LC))$ where:
  I = Row Subset of an element whose address to be found,
  J = Column Subset of an element whose address to be found,
  B = Base address,
  W = Storage size of one element store in an array(in byte),
  LR = Lower Limit of row/start row index of the matrix(If not given assume it as zero),
  LC = Lower Limit of column/start column index of the matrix(If not given assume it as zero),
  M = Number of column given in the matrix.

Column Major Order:
Address of $A[I][J] = B + W * ((I–LR) + N * (J–LC))$ where:
I = Row Subset of an element whose address to be found,
J = Column Subset of an element whose address to be found,
B = Base address,
W = Storage size of one element store in an array(in byte),
LR = Lower Limit of row/start row index of the matrix(If not given assume it as zero),
LC = Lower Limit of column/start column index of the matrix(If not given assume it as zero),
N = Number of rows given in the matrix.

- multi-D Array:
  Row Major Order:
  Address of
  $A[I][J][K] = B + W * (N * L(I - x) + L * (J - y) + (K - z))$
  where:
  B = Base Address (start address)
  W = Weight (storage size of one element stored in the array)
  N = Hight/Layer (total number of cells depth-wise)
  M = Row (total number of rows)
  L = Column (total number of columns)
  x = Lower Bound of Row
  y = Lower Bound of Column
  z = Lower Bound of Hight

Column Major Order:

Address of

$A[I][J][K] = B + W * (N * L * (I - x) + (J - y) + (K - z) * N)$

where:

B = Base Address (start address)

W = Weight (storage size of one element stored in the array)

N = Hight/Layer (total number of cells depth-wise)

M = Row (total number of rows)

L = Column (total number of columns)

x = Lower Bound of Row

y = Lower Bound of Column

z = Lower Bound of Hight

Application of arrays

Below are some applications of arrays.

- **Storing and accessing data**: Arrays are used to store and retrieve data in a specific order. For example, an array can be used to store the scores of a group of students, or the temperatures recorded by a weather station.
- **Sorting**: Arrays can be used to sort data in ascending or descending order. Sorting algorithms such as bubble sort, merge sort, and quicksort rely heavily on arrays.
- **Searching**: Arrays can be searched for specific elements using algorithms such as linear search and binary search.
- **Matrices**: Arrays are used to represent matrices in mathematical computations such as matrix multiplication, linear algebra, and image processing.

- **Stacks and queues**: Arrays are used as the underlying data structure for implementing stacks and queues, which are commonly used in algorithms and data structures.
- **Graphs**: Arrays can be used to represent graphs in computer science. Each element in the array represents a node in the graph, and the relationships between the nodes are represented by the values stored in the array.
- **Dynamic programming**: Dynamic programming algorithms often use arrays to store intermediate results of subproblems in order to solve a larger problem.

Below are some real-time applications of arrays:

- **Signal Processing**: Arrays are used in signal processing to represent a set of samples that are collected over time. This can be used in applications such as speech recognition, image processing, and radar systems.

- **Multimedia Applications**: Arrays are used in multimedia applications such as video and audio processing, where they are used to store the pixel or audio samples. For example, an array can be used to store the RGB values of an image.
- **Data Mining**: Arrays are used in data mining applications to represent large datasets. This allows for efficient data access and processing, which is important in real-time applications.
- **Robotics**: Arrays are used in robotics to represent the position and orientation of objects in 3D space. This can be used in applications such as motion planning and object recognition.

- **Real-time Monitoring and Control Systems**: Arrays are used in real-time monitoring and control systems to store sensor data and control signals. This allows for real-time processing and decision-making, which is important in applications such as industrial automation and aerospace systems.
- **Financial Analysis**: Arrays are used in financial analysis to store historical stock prices and other financial data. This allows for efficient data access and analysis, which is important in real-time trading systems.
- **Scientific Computing**: Arrays are used in scientific computing to represent numerical data, such as measurements from experiments and simulations. This allows for efficient data processing and visualization, which is important in real-time scientific analysis and experimentation.

Applications of Array in Java:

- **Storing collections of data**: Arrays are often used to store collections of data of the same type. For example, an array of integers can be used to store a set of numerical values.
- **Implementing matrices and tables**: Arrays can be used to implement matrices and tables. For example, a two-dimensional array can be used to store a matrix of numerical values.
- **Sorting and searching**: Arrays are often used for sorting and searching data. For example, the Arrays class in Java provides methods like sort() and binarySearch() to sort and search elements in an array.

- **Implementing data structures**: Arrays are used as the underlying data structure for several other data structures like stacks, queues, and heaps. For example, an array-based implementation of a stack can be used to store elements in the stack.
- **Image processing**: Arrays are commonly used to store the pixel values of an image. For example, a two-dimensional array can be used to store the RGB values of an image.

Sparse Matrices and their representations

A matrix is a two-dimensional data object made of n rows and m columns, therefore having total *mxn* values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

Why to use Sparse Matrix instead of simple matrix ?

- **Storage:** There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
- **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements.

- Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases.
- So, instead of storing zeroes with non-zero elements, we only store non-zero elements.
- This means storing non-zero elements with triples-
(Row, Column, value).

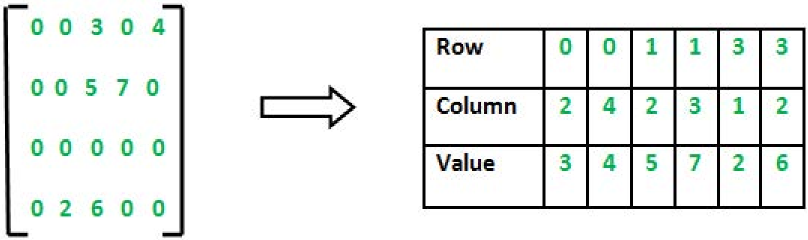
Sparse Matrix Representations can be done in many ways following are two common representations:

1. Array representation
2. Linked list representation

Method 1: Using Arrays

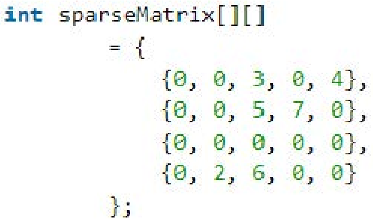
2D array is used to represent a sparse matrix in which there are three rows named as

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column)

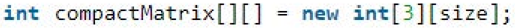


$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

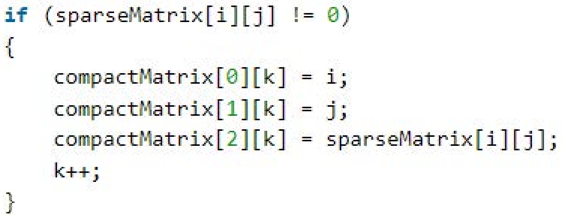
Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6



```
int sparseMatrix[][]
        = {
            {0, 0, 3, 0, 4},
            {0, 0, 5, 7, 0},
            {0, 0, 0, 0, 0},
            {0, 2, 6, 0, 0}
        };
```



```
int compactMatrix[][] = new int[3][size];
```

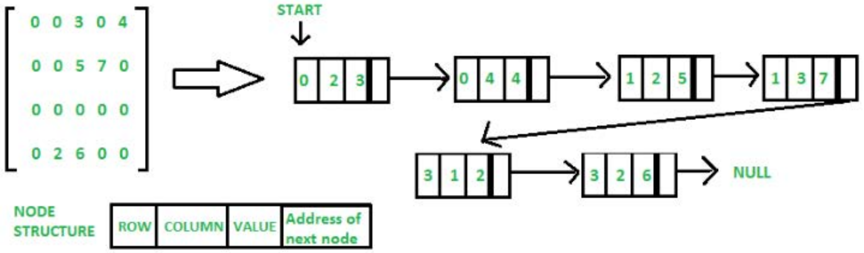


```
if (sparseMatrix[i][j] != 0)
{
    compactMatrix[0][k] = i;
    compactMatrix[1][k] = j;
    compactMatrix[2][k] = sparseMatrix[i][j];
    k++;
}
```

Method 2: Using Linked Lists

In linked list, each node has four fields. These four fields are defined as:

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column)
- **Next node:** Address of the next node



0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0
START
0 2 3
0 4 4
1 2 5
1 3 7
3 1 2
3 2 6
NULL
NODE STRUCTURE
ROW
COLUMN
VALUE
Address of next node

Arithmetic operations on matrices

- Addition of Matrix
- Subtraction of Matrix
- Scaler Multiplication of Matrix
- Multiplication of Matrix
- Transpose
- Inversion

Thank you

Please send your feedback or any queries to
subhash.chandra@galgotiasuniversity.edu.in