Data Structure

INTRODUCTION TO DATA STRUCTURE

UNIT - 1

ANKIT VERMA

ANKIT.VERMA@IITMJP.AC.IN
WWW.ANKITVERMA.CO.IN

DEPARTMENT OF INFORMATION TECHNOLOGY

Which Book To Follow ?

- Text Book
  - Data Structures, Schaum's Outlines, Seymour Lipschutz
- Reference Book
  - Data Structure Using C, Udit Aggarwal

Basic Terminology: Elementary Data Organization

Basic Terminology

- Data
  - Simply values or set of values.
- Data Items
  - Refers to a single unit of values.
  - Divided into Sub items:
    - Group Items
    - Elementary Items
- Fields, Records and Files
  - Collection of data are frequently organized into hierarchy of it.
- Entity
  - Something that has certain Attributes or Properties with values.

Basic Terminology

- Entity Set
  - Entries with similar attributes.
- Range
  - Each attribute of an entity set has range of values.
- Information
  - Data with given attributes or meaningful or processed data.
- Field
  - It is single elementary unit of information representing an attribute of an entity.
- Record
  - Collection of field values of a given entity.

Basic Terminology

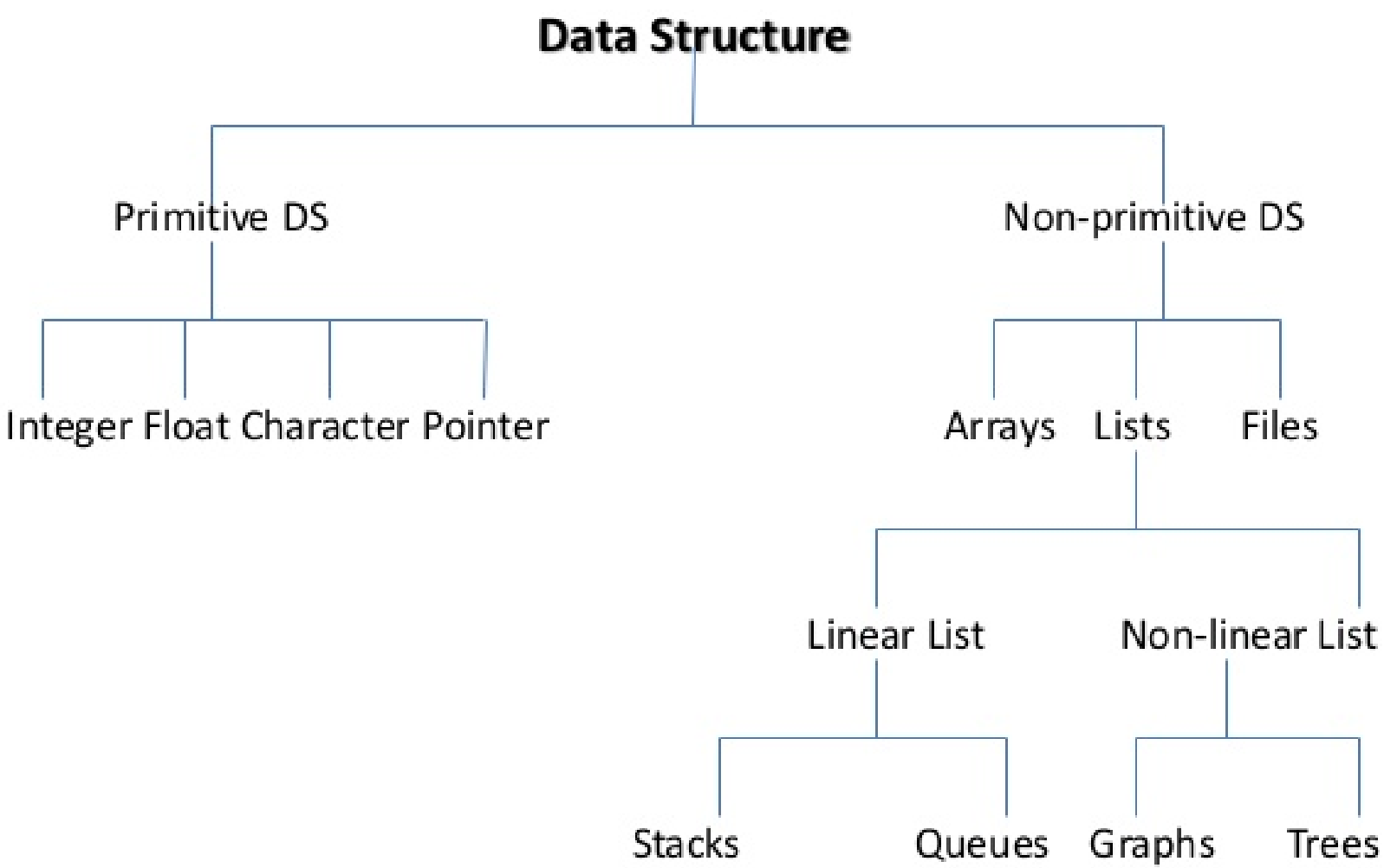
- File
  - Collection of records of entities in a given entity set.
- Primary Key
  - Value in certain field uniquely determine record in file.
  - Values in such field called Keys or Key Values.
- Fixed Length Record
  - All records contain same data items with same amount of space assigned to each data item.
- Variable Length Record
  - File records may contain different lengths.
- File or Database Management

Data Structure

Data Structure

- Logical or Mathematical model of a particular organization of data is called Data Structure.
- Choice of Data Model depends upon:
  1. It must be rich enough in structure to mirror the actual relationships of data in real world.
  2. The structure should be simple enough that one can effectively process the data when necessary.

Data Structure



Data Structure

- Primitive DS
  - These are basic structures and are directly operated upon by the machine instructions.
  - They have different representations on different computers.
    - E.g. Integer, Float, Character, Pointer
- Non-Primitive DS
  - These are more sophisticated data structures.
  - These are driven from Primitive DS.
    - E.g. Array, Lists, Files

Data Structure Classifications

- Data Structures are classified into:
  1. Linear
     - Elements form sequence or linear list.
     - Two ways of representation:
       1. Sequential memory location
          - E.g. Arrays
       2. Pointers or links
          - E.g. Linked List
  2. Non Linear
     - E.g. Tree & Graphs

Data Structure Operations

- Following four operations play a major role:
  - Traversing
    - Process each element in the list.
  - Searching
    - Finding location of element or record with given key.
  - Inserting
    - Adding new element to the list.
  - Deleting
    - Removing an element from the list.

Data Structure Operations

- Following two operations used in special situation:
  - Sorting
    - Arranging the elements in some type of order.
  - Merging
    - Combining two lists into single list.

Array

Linear Array

- Linear Array (LA) is a list of finite number n of homogeneous data elements such that:
  - Elements referenced respectively by an Index Set.
  - Elements stored respectively in successive memory locations.

Length / Size of Array = UB – LB + 1

- Upper Bound (UB)
  - It is the largest index.
- Lower Bound (LB)
  - It is the smallest index.

Linear Array Notation

- Subscript Notation
  - $A_1, A_2, A_3, \ldots\ldots, A_n$
- Parentheses Notation
  - A(1), A(2), A(3), ..., A(n)
- Bracket Notation
  - A[1], A[2], A[3], ..., A(n)

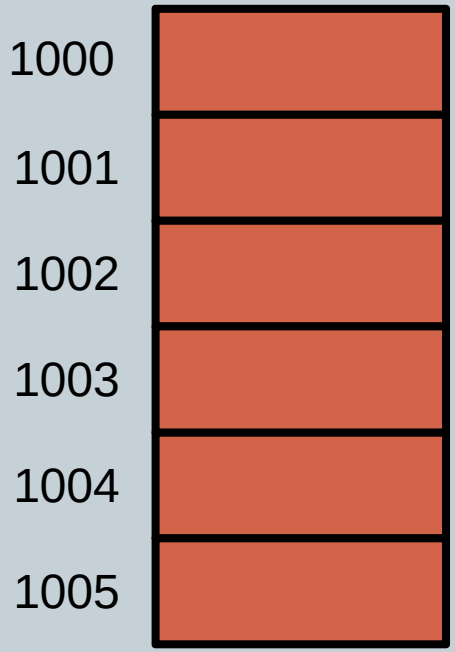
Linear Array Memory Allocation

- Static Memory Allocation
  - Memory allocated at Compile time.
  - Size of Array fixed before Execution.
- Dynamic Memory Allocation
  - Memory allocated at Run time.
  - Size of Array may vary.

Linear Array Representation

- LOC (LA [K]) = Address of element LA [K] of array LA
- LOC (LA [K]) = Base (LA) + w (K – Lower Bound)

- Base (LA) is first element of LA.
- K is any value.
- W is number or words per memory cell.
- LA is Linear Array.



Traversing Linear Array

- ALGORITHM: Traversing Linear Array
  - Let LA is linear array with lower bound LB and upper bound UB. Algorithm traverses LA applying an operation PROCESS to each element of LA.
    1. Set K :=LB.
    2. Repeat Steps 3 and 4 while K <=UB
    3. Apply PROCESS to LA[K].
    4. Set K := K + 1.
    5. Exit

Traversing Linear Array

- ALGORITHM: Traversing Linear Array
  - This algorithm traverses a linear array LA with lower bound LB and upper bound UB.
    1. Repeat for K = LB to UB:
       Apply PROCESS to LA[K].
    2. Exit

Inserting Into Linear Array

- ALGORITHM: INSERT (LA, N, K, ITEM)
  - Here LA is linear array with N elements and K is positive integer such that K<=N. This algorithm inserts an element ITEM into Kth position in LA.
    1. Set J := N.
    2. Repeat Steps 3 and 4 while J >= K.
    3. Set LA[J + 1] := LA[J].
    4. Set J := J – 1.
    5. Set LA[K] := ITEM.
    6. Set N := N + 1.
    7. Exit

Deleting From Linear Array

- ALGORITHM: DELETE (LA, N, K, ITEM)
  - Here LA is linear array with N elements and K is positive integer such that K<=N. This algorithm deletes the Kth element from LA.
    1. Set ITEM := LA[K].
    2. Repeat for J := K to N – 1:
       Set LA[J] := LA[J + 1].
    3. Set N := N – 1.
    4. Exit

Multidimensional Arrays

- One Dimensional Array
  - Linear Array (LA)
- Two Dimensional Array
  - m X n Array is collection of m*n data elements such that each element is specified by a pair of integers (j, k) called subscripts.

    where 1 <= j <= m and 1 <= k <=n
  - Element of Array with first subscript j and second subscript k denoted by $A_{j,k}$ or A[j, k].
  - In Mathematics 2D Array is called Matrices.
  - In Business Applications 2D Array is called Tables.
  - 2D Array also called Matrices Arrays.
  - Size of Array = m X n

Multidimensional Arrays

- Non Regular Array
  - Multidimensional array in which lower bounds are not 1.
- Regular Array
  - Multidimensional array in which lower bounds are 1.

- n-Dimensional Array
  - m_1 X m_2 X m_3 X ..... X m_n is collection of m_1*m_2*m_3*..... M_n elements in which each element is specified by a list of n integers such as K_1, K_2, ..., K_n.

    where 1 <= K_1 <=m_1, ..., 1 <= K_n <= m_n
  - Element of Array with subscript K_1, K_2, ..., K_n is denoted by $A_{K1, K2, ..., Kn}$ or A[K_1, K_2, ..., K_n]

Representation of 2D Array in Memory

- 2D Array m X n is represented as m Rows and n Columns by m*n Sequential Memory Locations.
- Column-major Order
- Row-major Order

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Row-major

1	2	3	4	5	6	7	8	9	10	11	12

Column-major

1	5	9	2	6	10	3	7	11	4	8	12

Representation of 2D Array in Memory

- Column-major Order

(1,1)	
(2,1)	Column1
(3,1)	
(1,2)	
(2,2)	Column2
(3,2)	
(1,3)	
(2,3)	Column3
(3,3)	
(1,4)	
(2,4)	Column4
(3,4)	

- Row-major Order

(1,1)	
(1,2)	Row1
(1,3)	
(1,4)	
(2,1)	Row2
(2,2)	
(2,3)	
(2,4)	
(3,1)	Row3
(3,2)	
(3,3)	
(3,4)	

Representation of 2D Array in Memory

- Column-major Order
  - LOC(A[j,k]) = Base(A) + w[m(k – 1) + (j – 1)]
- Row-major Order
  - LOC(A[j,k]) = Base(A) + a[n(j – 1) + (k – 1)]
- W is word per memory location for array
- Base(A) is address of first element A[1,1]

Address Calculation

- Calculate the Address of X[4, 3] in a Two Dimensional Array X[1.....5, 1.....4] stored in Row Major order in the main memory. Assume the Base Address to be 1000 and that each element requires 4 bytes of storage.
- Solution:

Base Address B = 1000

Word Size W = 4 bytes

No. of Columns n = 4

First Row Number L1 = 1

First Column Number L2 = 1

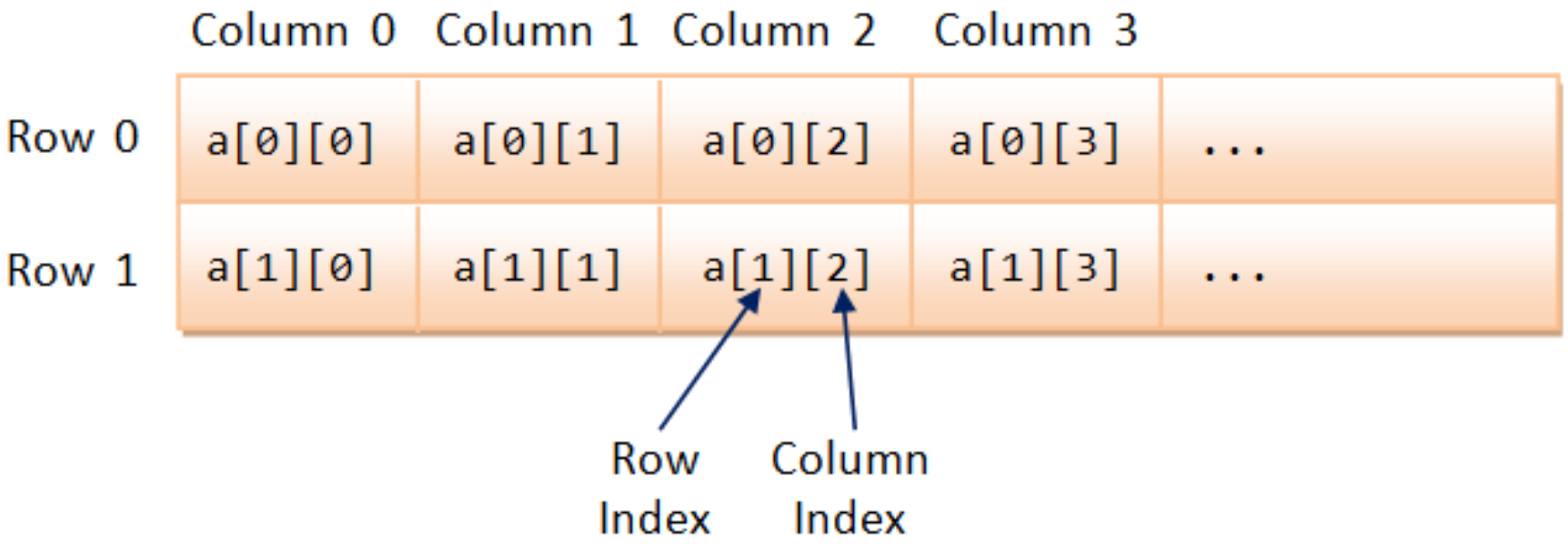
Address X[i, j] = B + W[n(i – L1) + (j – L2)]

Given i = 4, j = 3

Address X[4, 3] = 1000 + 4[4(4 – 1) + (3 – 1)] = 1056

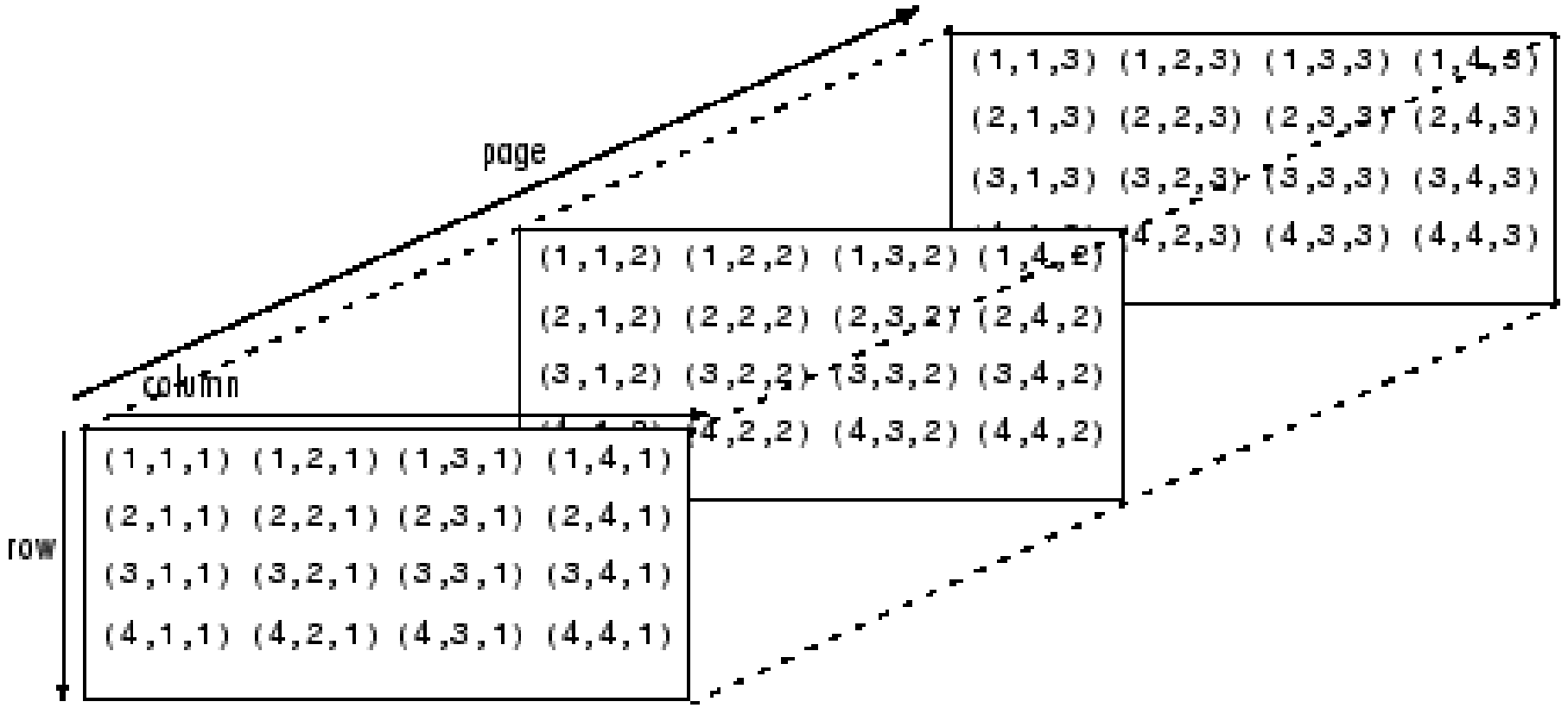
Representation of 2D Array

- 2 Dimension Array Representation



Representation of nD Array

- n Dimension Array Representation



Matrices

Matrices

- n-Element Vector
  - V = (V1, V2, V3, …, Vn)
- m X n Matrices
  - m Rows and n Columns
  - One Row of matrix may be viewed as Vector.
  - Vector may be viewed as Matrix with only one Row.
- Square Matrix / n-Square Matrix
  - Same number of Rows and Columns.
- Diagonal / Main Diagonal Matrix
  - Consist only Diagonal elements A11, A22, ….., Ann

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Matrix Multiplication

- ALGORITHM: MATMUL (A, B, C, M, P, N)
  - Let A be an M X P matrix array and B be a P X N matrix array. This algorithm stores the product of A and B in an M X N matrix array C.

1. Repeat Steps 2 to 4 for I = 1 to M
2. Repeat Steps 3 and 4 for J = 1 to N
3. Set C [I, J] := 0
4. Repeat for K =1 to P:

   C [I, j] := C [I, J] + A [I, K] * B [K, J]
5. Exit

Sparse Matrices

- Dense Matrices
  - Few elements are zero.
- Sparse Matrices
  - Matrices with relatively high proportion of zero entries.
  - Many elements are zero.

$$\begin{pmatrix} 0 & 3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 8 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 4 & 0 \\ 0 & 1 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \end{pmatrix}$$

Forms of Sparse Matrices

- Diagonal Matrix
  - Non Zero elements are stored in leading diagonal of matrix.

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -7 \end{bmatrix}$$

- Tridiagonal Matrix
  - Non Zero elements are placed on or below or above leading diagonal.

$$\begin{pmatrix} 1 & 4 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 0 & 2 & 3 & 4 \\ 0 & 0 & 1 & 3 \end{pmatrix}$$

- Lower Triangular Matrix
  - Non Zero elements are placed below leading diagonal.

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 2 & -1 & 0 & 0 \\ 1 & -2 & 2 & 0 \\ 1 & 3 & 4 & 2 \end{bmatrix}$$

- Upper Triangular Matrix
  - Non Zero elements are placed above leading diagonal.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

Storage of Sparse Matrices

- If user stores entire matrix including zero elements then there is wastage of storage space.
- Significant Storage & Computational Savings can be achieved in two ways:
  - Array Representation
  - Linked-list Representation

Storage of Sparse Matrices

- Array Representation
  - Only Non Zero elements are stored.
  - Non Zero element in Sparse Matrix is represented as:

    (Row, Column, Value)
  - Two dimension array containing 3 Columns can be used.

0	0	0	0	3	0
0	9	0	0	0	2
0	0	1	8	0	0
0	0	0	0	4	0
3	0	0	0	0	0

Row	Column	Non-zero
0	4	3
1	1	9
1	5	2
2	2	1
2	3	8
3	4	4
4	0	3

  - Limitation:
    - Need to know number of Non Zero elements in Matrix.
    - Not possible to predict size of resultant matrix before hand.

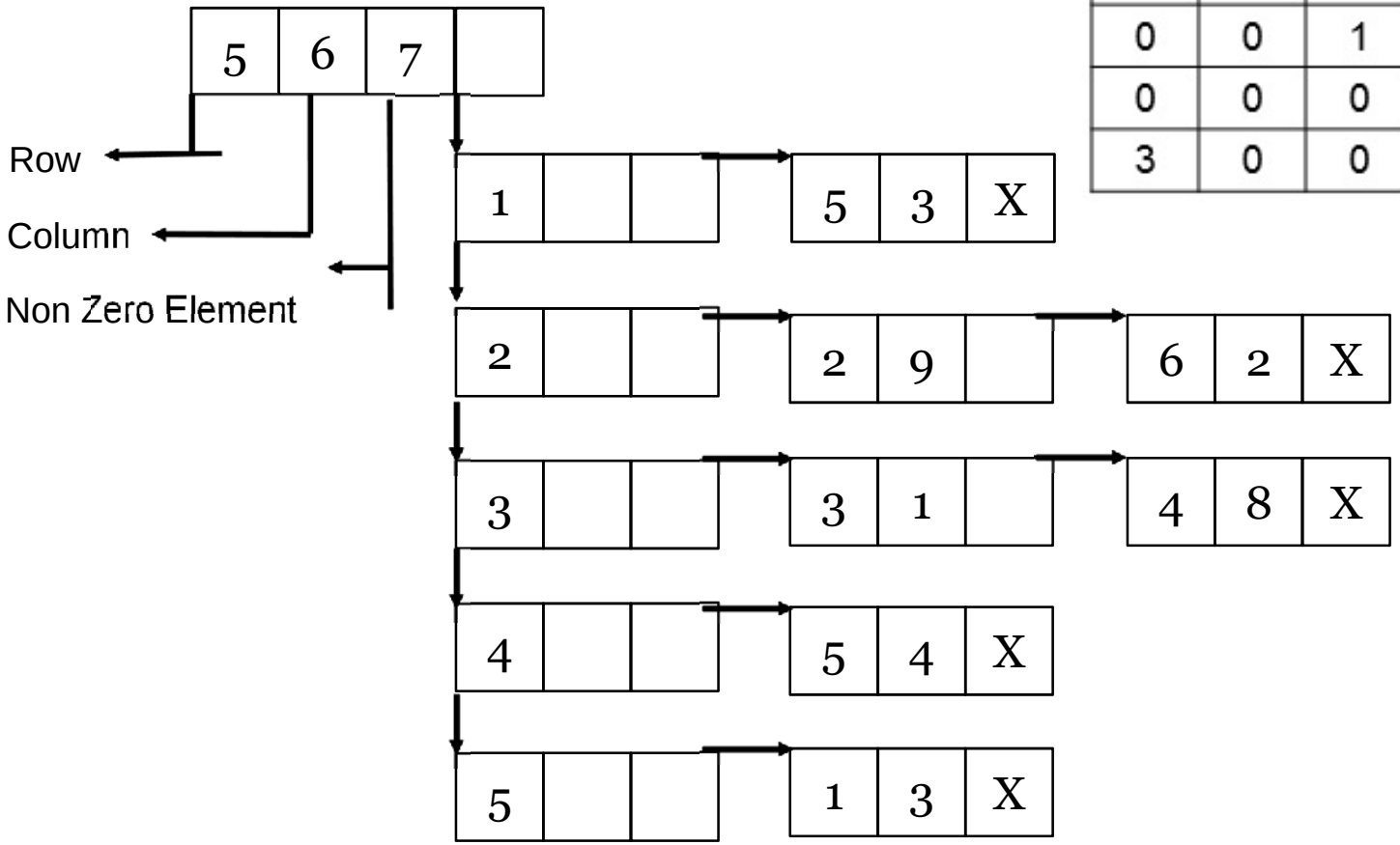
Storage of Sparse Matrices

- Linked List Representation
  - In linked list representation a sparse list is maintained for each column as well as each row of matrix.
  - If Matrix is of size 4 X 4, then 4 lists for 4 column and 4 lists for 4 rows.
  - Node store information about Non Zero element, i.e. Row Number, Column Number and Value of Non Zero Element.
  - Head node for Column List stores Column Number, a pointer to node which comes first in the Column and a pointer to next column Head Node.
  - Head node for Row List stores, a pointer to node, which comes first in Row List and a pointer to next Row Head node.

Storage of Sparse Matrices

- Linked List Representation

0	0	0	0	3	0
0	9	0	0	0	2
0	0	1	8	0	0
0	0	0	0	4	0
3	0	0	0	0	0

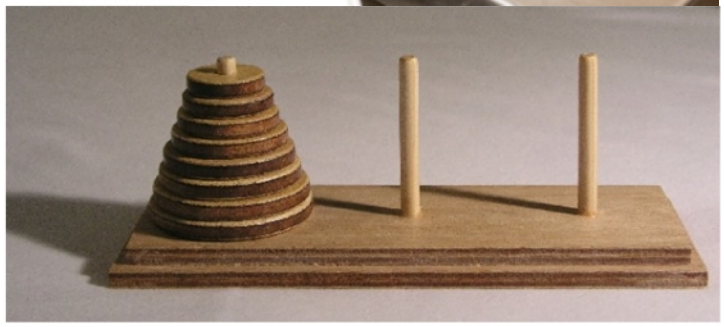


Stack

Stack

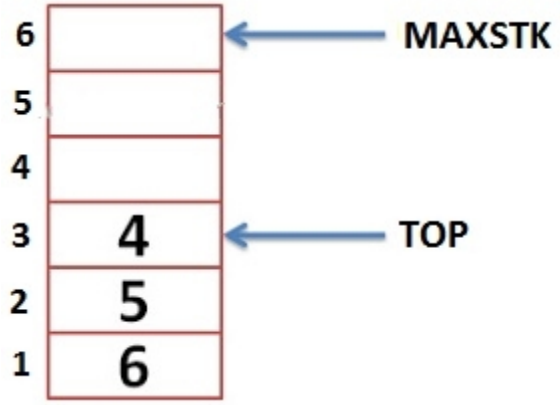
- In certain situations Insertion and Deletion only possible at Beginning or End of List, but not in Middle.
  - E.g. Stack and Queue
- Stack is a Linear structure in which Items may be Added or Removed only at one end.
- Last Item to be Added to Stack is first item to be Removed.
- Also called Last-In-First-Out (LIFO) Lists, Piles or Push-Down Lists.

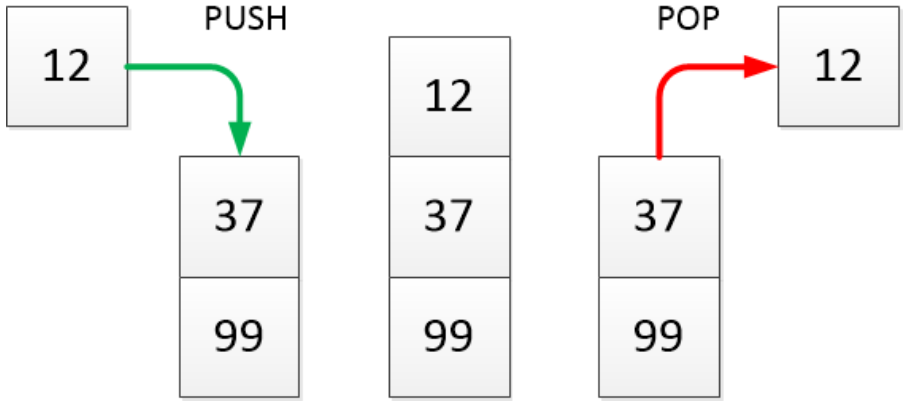
Stack Examples



Stack Basic Operations

- Stack is a list of elements in which an element may be Inserted or Deleted only at one end, called Top of Stack.
- Elements are Removed from Stack in Reverse order in which they are Inserted.
- Stack Basic Operations:
  - Push
    - Insert an element into Stack.
  - Pop
    - Delete an element from Stack.





Stack Basic Operations : PUSH

- ALGORITHM: PUSH (STACK, TOP, MAXSTK, ITEM)
  - This procedure pushes an ITEM onto a Stack.
    1. If TOP = MAXSTK, then: Print OVERFLOW, and Return.
    2. Set TOP := TOP + 1.
    3. Set STACK [TOP] := ITEM.
    4. Return.

Stack Basic Operations : POP

- ALGORITHM: POP (STACK, TOP, ITEM)
  - This procedure deletes the top element of STACK and assigns it to the variable ITEM.
    1. If TOP = 0, then: Print UNDERFLOW, and Return.
    2. Set ITEM := STACK [TOP].
    3. Set TOP := TOP – 1.
    4. Return.

Polish Notation

- Infix Notation
  - Most common Arithmetic operations place operator symbol between two operands.
  - Example:
    - A + B
    - (A + B) * C
- Polish / Prefix Notation
  - Named after Polish mathematician Jan Lukasiewicz.
  - Operator symbol is placed before its two operands.
  - Example:
    - + A B
    - * + A B C

Polish Notation

- Reverse Polish / Postfix / Suffix Notation
  - Operator symbol is placed after its two operands.
  - Example:
    - A B +
    - A B +C *

Infix To Polish Notation

- (A + B) * C = [+ A B] * C = * + A B C
- A + (B * C) = A + [* B C] = + A * B C
- (A + B) / (C – D) = [+ A B] / [- C D] = / + A B – C D

Evaluation of Postfix Expression

- Evaluate Arithmetic expression P as Postfix Expression:

P: 5, 6, 2, +, *, 12, 4, /, -,)

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

SYMBOL	SCANNED	STACK
(1)	5	5
(2)	6	5, 6
(3)	2	5, 6, 2
(4)	+	5, 8
(5)	*	40
(6)	12	40, 12
(7)	4	40, 12, 4
(8)	/	40, 3
(9)	-	37
(10)	)	

Evaluation of Postfix Expression

- Evaluate Arithmetic expression P as Postfix Expression:

P: 5, 9, 8, +, 4, 6, *, +, 7, -, * ,)

1 2 3 4 5 6 7 8 9 10 11 12

SYMBOL	SCANNED	STACK
(1)	5	5
(2)	9	5, 9
(3)	8	5, 9, 8
(4)	+	5, 17
(5)	4	5, 17, 4
(6)	6	5, 17, 4, 6
(7)	*	5, 17, 24
(8)	+	5, 41
(9)	7	5, 41, 7
(10)	-	5, 34
(11)	*	170
(12)	)	

Evaluation of Postfix Expression

- ALGORITHM: Evaluation of Postfix Expression
  - This Algorithm finds the VALUE of an arithmetic expression P written in postfix notation.
    1. Add a right parenthesis ")" at the end of P.
    2. Scan P from left to right and repeat Steps 3 & 4 for each element of P until the sentinel ")" is encountered.
    3. If an operand is encountered, put in on STACK.
    4. If an operator Θ is encountered then:
       a) Remove the two top elements of STACK, where A is top element and B is next-to-top element.
       b) Evaluate B Θ A.
       c) Place the result of (b) back on STACK.
    5. Set VALUE equal to the top element on STACK.
    6. Exit

Transform Infix into Postfix

- Transform Q into its equivalent Postfix Expression:

Q: A + (B * C – (D / E † F) * G) * H)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Q: A + (B * C – (D / E † F) * G) * H)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

SYMBOL	SCANNED	STACK	EXPRESSION
1	A	(	A
2	+	(+	A
3	(	(+ (	A
4	B	(+ (	A B
5	*	(+ (*	A B
6	C	(+ (*	A B C
7	–	(+ (–	A B C *
8	(	(+ (– (	A B C *
9	D	(+ (– (	A B C * D
10	/	(+ (– (/	A B C * D
11	E	(+ (– (/	A B C * D E
12	†	(+ (– (/ †	A B C * D E
13	F	(+ (– (/ †	A B C * D E F
14	)	(+ (–	A B C * D E F † /
15	*	(+ (– *	A B C * D E F † /
16	G	(+ (– *	A B C * D E F † / G
17	)	(+	A B C * D E F † / G * –
18	*	(+ *	A B C * D E F † / G * –
19	H	(+ *	A B C * D E F † / G * – H
20	)		A B C * D E F † / G * – H * +

Transform Infix into Postfix

- ALGORITHM: POLISH (Q, P)
  - Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.
    1. Push "(" onto STACK and add ")" to the end of Q.
    2. Scan Q from left to right and repeat Steps 3 to 6 for each element of Q until the STACK is empty.
    3. If an operand is encountered, add it to P.
    4. If a left parenthesis is encountered, push it onto STACK.
    5. If an operator Θ is encountered then:
       a) Repeatedly pop from STACK and add to P each operator on top of STACK which has same precedence as or higher precedence than Θ.
       b) Add Θ to STACK.
    6. If a right parenthesis is encountered then:
       a) Repeatedly pop from STACK and add to P each operator on top of STACK until a left parenthesis is encountered.
       b) Remove the left parenthesis.
    7. Exit

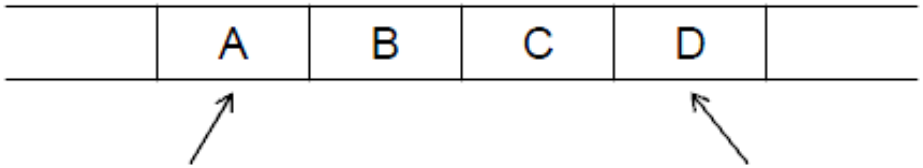
Queue

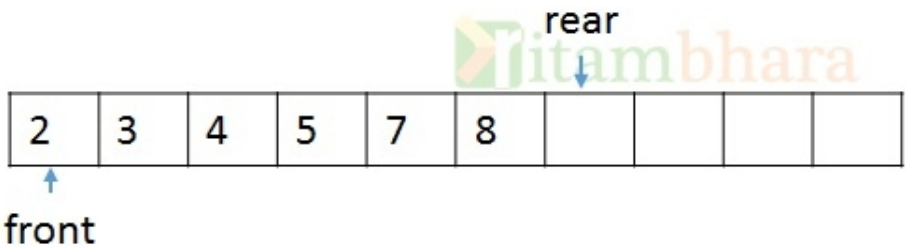
Queue

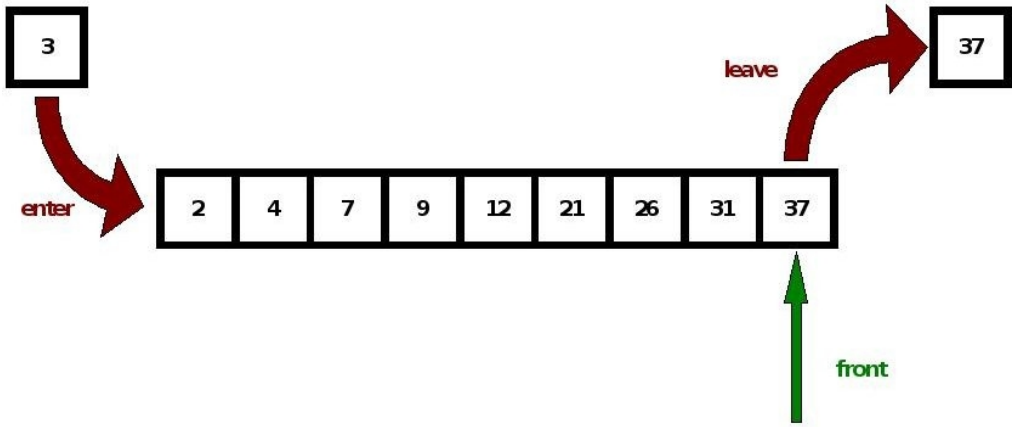
- Queue is a Linear List of elements in which Deletion can place only at one end called Front and Insertion can take place only at other end called Rear.
- Also called First In First Out (FIFO).
- Example:
  - Automobiles waiting to pass through.
  - Programs with same priority from a Queue are waiting to be executed.

Queue Examples

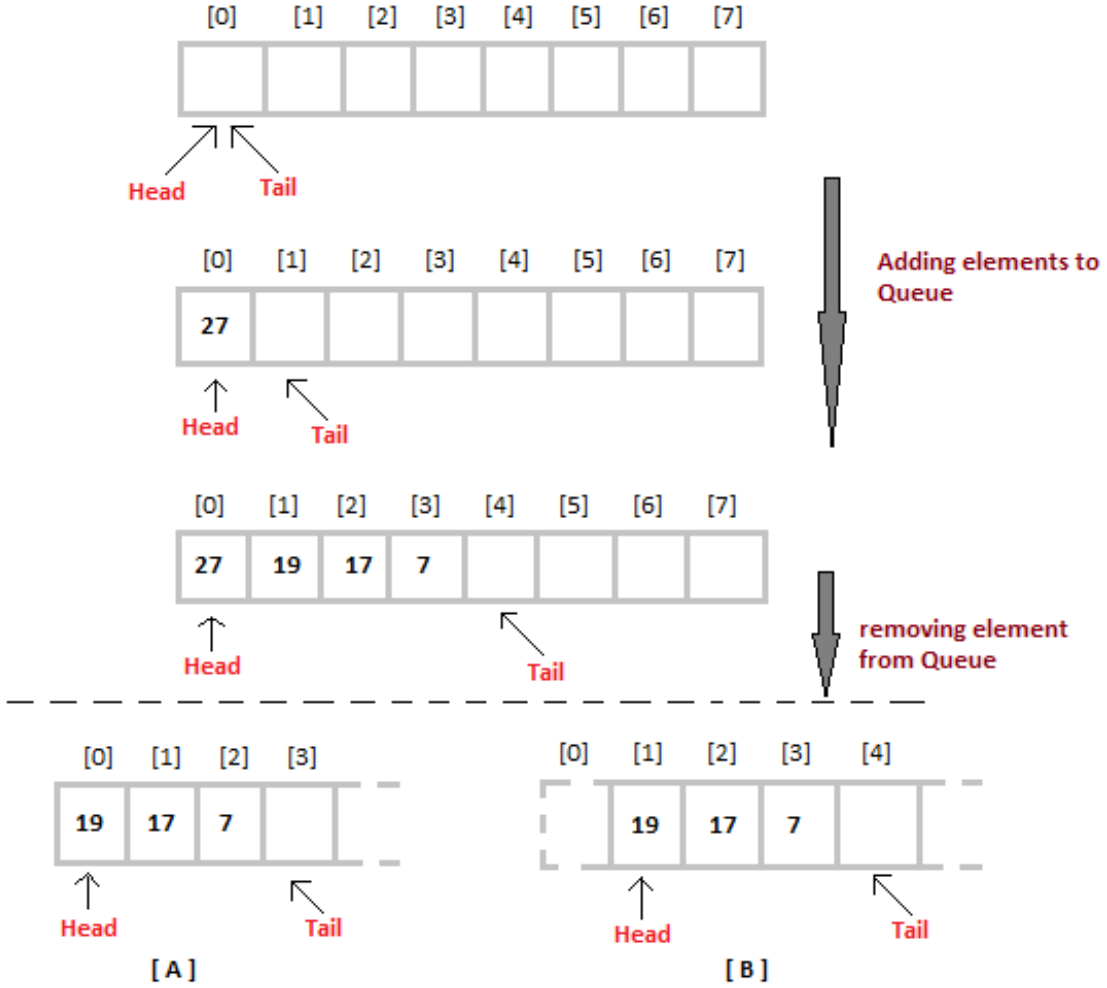
Representation of Queue







Insertion & Deletion in Queue



Insert Element in Queue

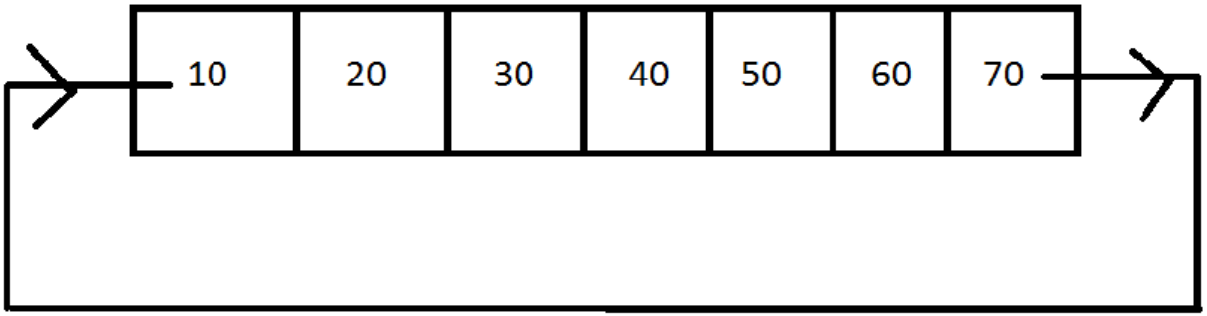
- ALGORITHM: QINSERT (QUEUE, N, FRONT, REAR, ITEM)
  - This procedure inserts an element ITEM into a queue.
    1. If FRONT = 1 and REAR = N, or if FRONT = REAR + 1 then:
       Write: OVERFLOW, and Return.
    2. If FRONT := NULL, then:
       Set FRONT :=1 and REAR := 1.
       Else if REAR := N, then:
       Set REAR := 1.
       Else:
       Set REAR := REAR +1.
    3. Set QUEUE [REAR] := ITEM.
    4. Return.

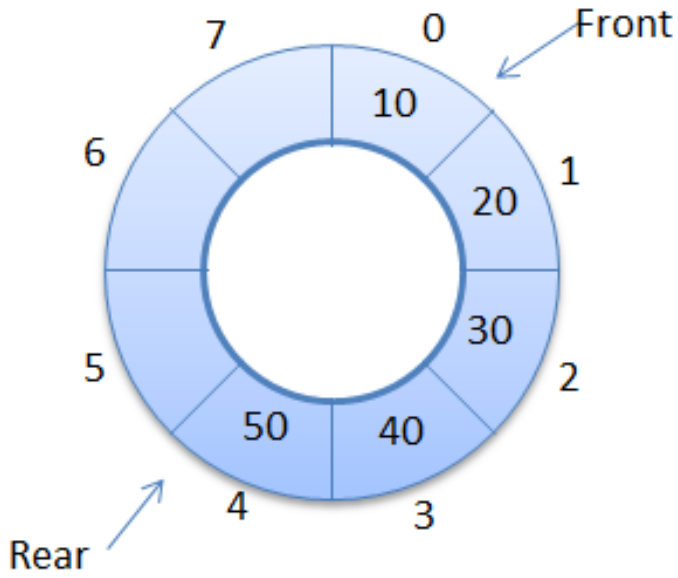
Delete Element in Queue

- ALGORITHM: QDELETE (QUEUE, N, FRONT, REAR, ITEM)
  - This procedure deletes an element from a queue and assign it to the variable ITEM.

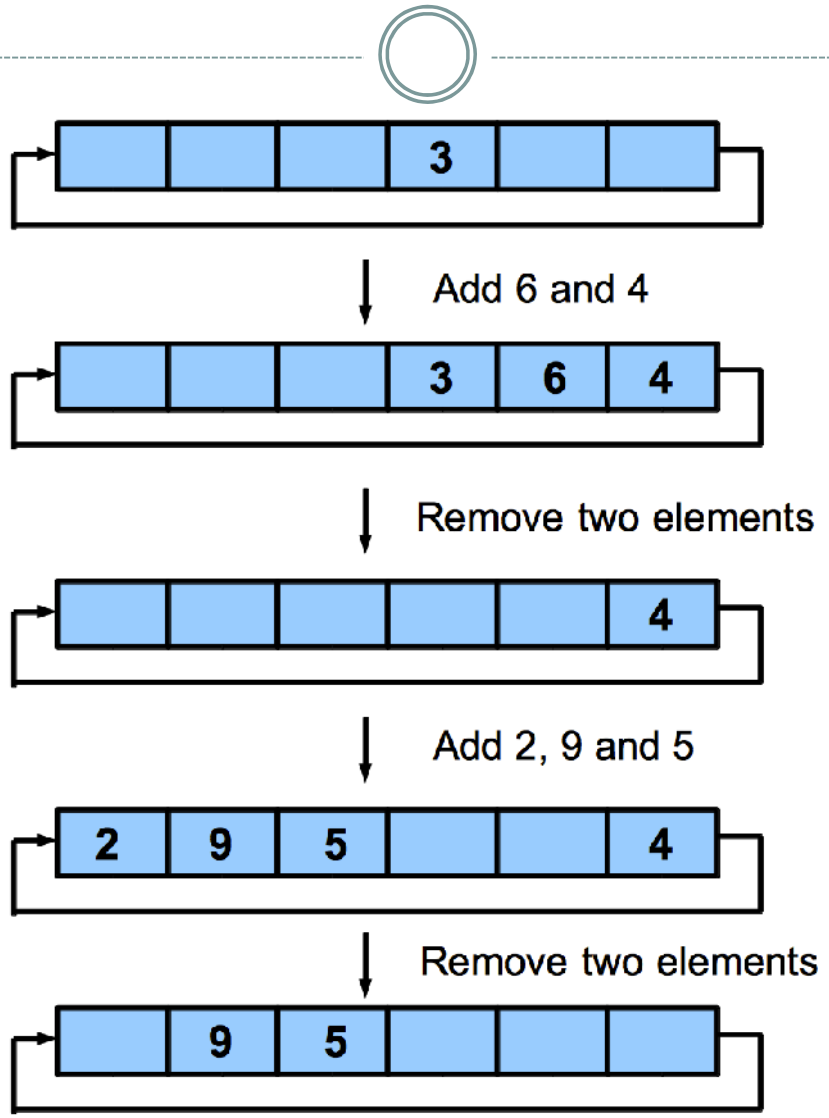
1. If FRONT := NULL, then:
   Write: UNDERFLOW, and Return.
2. Set ITEM := QUEUE [FRONT].
3. If FRONT := REAR, then:
   Set FRONT := NULL and REAR := NULL.
   Else if FRONT := N, then:
   Set FRONT := 1.
   Else:
   Set FRONT := FRONT +1.
3. Return.

Circular Queue





Insertion & Deletion in Circular Queue

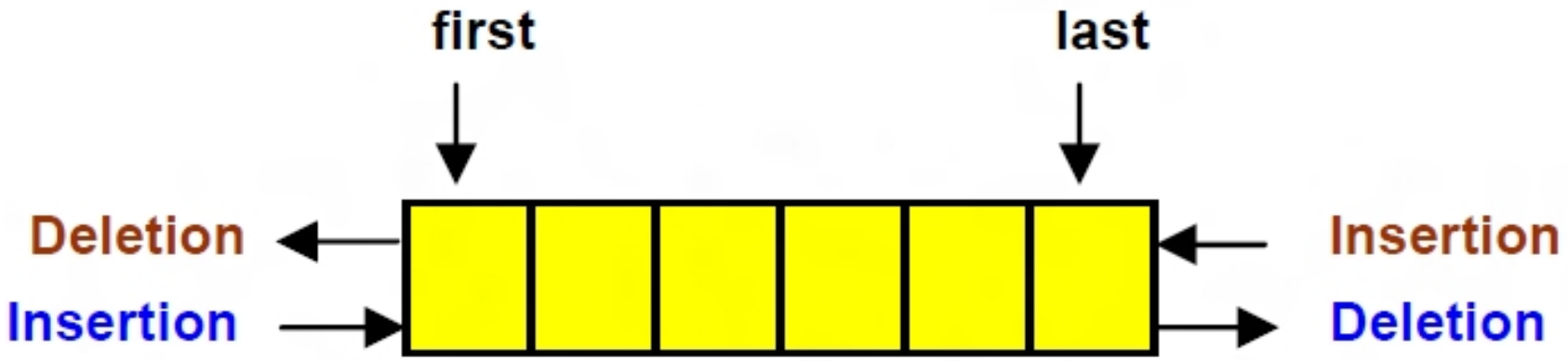


DEQUES

- DEQUES is a Linear List in which elements can be Added or Removed at either end but not in middle.
- Also called Double Ended Queue.
- It is Circular Array DEQUE with pointers LEFT and RIGHT.
- Two Variants of DEQUE:
  - Input Restricted DEQUE
    - Allow Insertion at only one end, but allow Deletions at both ends.
  - Output Restricted DEQUE
    - Allow Deletion at only one end, but Insertions at both ends.

DEQUES


Deletion
Insertion
Insertion
Deletion
Front
Rear


first
last
Deletion
Insertion
Insertion
Deletion

Priority Queues

- Priority Queue is collection of elements such that each element has been assigned a Priority and order in which elements are Deleted and Processed comes from following rules:
  - An element of Higher Priority is Processed before any element of Lower Priority.
  - Two elements with Same Priority are Processed according to order in which they were added to Queue.
- Example:
  - Timesharing System

THANKYOU

DOUBTS

PRESENTATION BY:
ANKIT VERMA
(IT DEPARTMENT)