Basic Concepts of Data Structures

❖ 1.1 Data Structure Basic Concepts

Data structure is a method of organizing large amount of data more efficiently so that any operation on that data becomes easy.

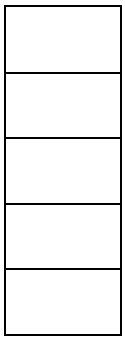
Any processed data is also known as information.

Storage Representation:

Data is stored in RAM or ROM.

Cell is a memory location, which is used to store elements of data items.

Cell stores 8 bit value.



❖ 1.2 Types of Data Structure

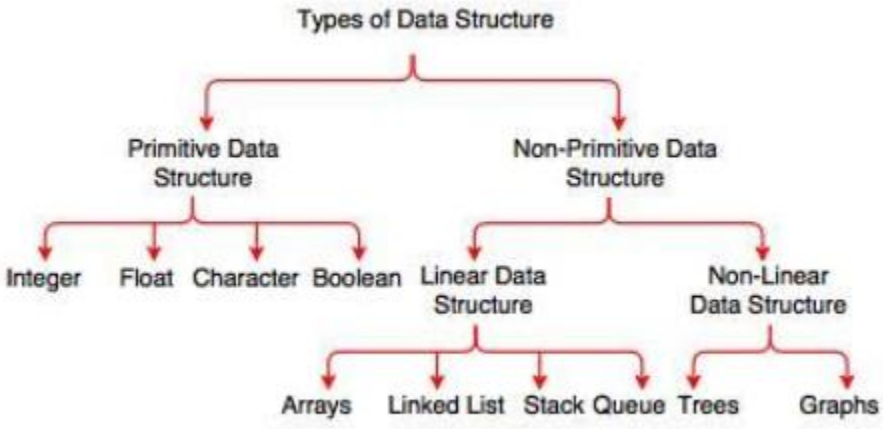


Fig. Types of Data Structure

Primitive Data Structure: The Data structures that are directly processed by machine using its instructions are known as primitive data structure.

- Following are the primitive data structure:

Integer:

- Integer represents numerical values.
- It can be + or -.
- No of student in a class.
- The sign - magnitude method and 1's complement method are used to represent integer numbers.
- Ex: 1000

Real:

- The number having fractional part is called real number.
- Ex: 123.45
- Real numbers are also stored in scientific format.

Character:

- Character data structure is used to store nonnumeric information.
- It can be letters [A-Z], [a-z], digits and special symbols.
- A character is represented in memory as a sequence bits.
- Two most commonly known character set supported by computer are ASCII and EBCDIC

Logical:

- A logical data item is a primitive data structure that can assume the values of either “true” or “false”.

Pointer:

- Pointer is a variable which store the address of another variable.
- It provides faster insertion and deletion of elements.

Non Primitive Data Structure: The data structures that are not directly processed by machine using its instructions are known as non primitive data structure.

Non Primitive Data Structure is classified into two categories:

- Linear Data structure

1) Non linear Data structure

1) Linear Data Structure:

Logical relation among the data item is linear.

- Data is stored sequentially in memory location.
- Examples of the linear data structure are:

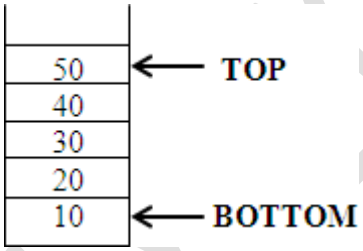
(A) Array (B) Stack (C) Queue (D) Linked List

(A) **Array:** Set of data item using same data type.

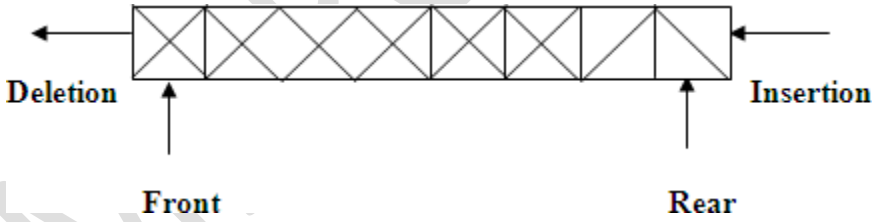
Set of fixed number of item.

Insert and delete operation are time consuming, searching and sorting are fast.

(B) **Stack**: A stack is a linear list in which insertion and deletion operations are performed at only one end of the list. It is LIFO (Last In First Out).



C) **Queue**: A queue is a linier list in which insertion is performed at one end called rear and deletion is performed at another end of the list called front.



It is FIFO (First In First Out).

(D) **Linked list:** A linked list is a collection of nodes.

Each node has two fields:

(i) Information

(ii) Address, which contains the address of the next node.

2) Non Linear data structure:

- Logical relation among the data item is not linear.
- Data is not stored sequentially.
- Examples of the non linier data structure are:

(A)Tree (B) Graph

(A) Tree: It consists of nodes connected by edge. Nodes represent data items. The node represent by circle.

(B) Graph: This data structure contains relationship between pairs of elements. Graphs are many types.

1. Directed graph
2. Undirected graph
3. Mixed Graph
4. Weighted graph

- ✓ There are various types of Data Structure which are classified according to several ways, such are
  - ✓ Linear Data Structure: Logical relation among the data item is linear. Ex: Array,Stack,Queue

Non Linear Data Structure: Logical relation among the data item is not linear.

  - ✓ Ex: Tree, Graph
  - ✓ Static Data structure: The size of the Data Structure is fixed and not changed during execution of program. Ex: Array
  - ✓ Dynamic Data Structure: It is created using dynamic memory location. For that, we use a pointer variable and its size is allocated during the execution of the program. We can increase or decrease the size of the data structure easily. Ex: Linked List
  - ✓ Homogeneous Data Structure: It stores all data of same data type.

Ex: Array

- ✓ Non-Homogeneous Data Structure: It contains different data type. Ex:Structure,class

❖ 1.3 Primitive and non-primitive data structures

Primitive Datatypes	Non-Primitive Datatypes
Primitive Datatypes are pre-defined by the language itself.	Non-Primitive Datatypes need to be defined by the user.
When declared primitive data is stored in the stack.	The reference variables are stored in the stack but the original object is in the heap.
When a copy is created for the data, a completely separate allocation is done for the copy.	Copies here create new reference variables but they both point to the same object in the heap.
Thus, the changes made to the copy are not reflected in the original.	Here however the changes made to the copy trace back to the original.
The primitive data structure will contain some value, i.e., it cannot be NULL.	The non-primitive data structure can consist of a NULL value.
The size depends on the type of data structure.	In the case of a non-primitive data structure, size is not fixed.
The primitive data structure can be used to call the methods.	Non-primitive data structure cannot be used to call the methods.

❖ 1.4 Introduction to Algorithms

Algorithm: It is a finite sequence of instructions necessary to solve problem using computer.

Characteristics of Algorithm

- Input: Inputs are provided before an algorithm.
- Output: Algorithm must have one or more outputs.
- Each instruction is clear and performs a specific action.
- Total no of repetition must be finite.
- Total time to carry out all the steps must be finite Key

❖ 1.5 Key Features of an Algorithm:

1. **Name of Algorithm**: Every Algorithm is given an identifying name, written in capital letters.
2. **Steps:** The algorithm is made of sequence of steps.

3. **Assignment Statements**: The assignment statement is indicated by arrow. Ex:x ←10
4. **If Statements:**
    a) If (condition)
    then
    b) If (condition)
    then
    else
5. **Repeat Statement**
    a) Repeat while(condition)
    b) Repeat thru step No for variable name=1,2,…N
    c) Case Statement

Select case

Case value 1:

Case value 2:

Case value N:

Default:

6. **GOTO Statement**: It is used for unconditional transfer of controls.
7. **Exit Statement**: It is used to terminate an algorithm.

❖ 1.6 Analysis Terms (for the definitions purpose only)

Time complexity: Time complexity is defined as the amount of time taken by an algorithm to run, as a function of the length of the input.

Complexity: Complexity of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data.

Space Complexity: It is a function describing the amount of memory an algorithm takes in terms of the amount of input to the algorithm.

Asymptotic Notation: Is a way of expressing the cost of an algorithm.

Big oh notation (O): It is define as upper bound and upper bound on an algorithm is the most amount of time required (the worst case performance). Big oh notation is used to describe asymptotic upper bound.

In computer science, big O notation is used to classify algorithms according to how their run

time or space requirements grow as the input size grows.

Big Omega Notation (Ω): It is defined as lower bound and lower bound on an algorithm is the least amount of time required (the most efficient way possible, in other words best case).Ω notation provides asymptotic lower bound

Big Theta Notation (Θ): It is define as tightest bound and tightest bound is the best of all the worst case times that the algorithm can take.
Worst Case Complexity: Slowest time to complete with chosen input.

When the algorithm takes maximum no of steps on any instance of size n. Example: In linear search, If searching element is at last position Or In ascending order sorting, if all the elements are in descending order then time taken by the algorithm is more.

Best Case Complexity: Fastest time to complete with chosen input.

When the algorithm takes minimum no of steps on any instance of size n. Example: In linear search, If searching element is at First position Or In ascending order sorting, if all the elements are in ascending order then time taken by the algorithm is less.

Average case: In average case analysis, we take all possible inputs and calculate the computing time for all of the inputs.

❖ 1.7 Array

- One dimensional array can be declared as:

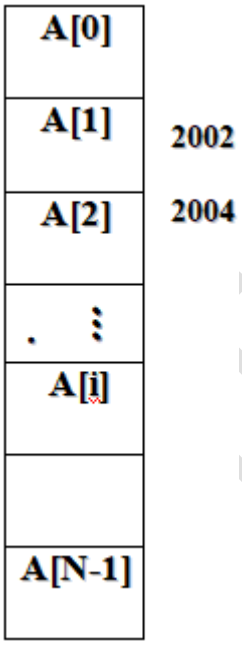
  Example: - int a [20].

Storage Representation of One Dimensional Array:

- Representation of an array in memory location is shown below:

Base Address (L0) 2000

L0 + (I - L) * s



Address of A [i] = Loc (A[i]) = Base + (I – L) * s

Thus location (Address) of A[i] in the array is calculated as:

- L0 is the starting (Base) address or address of first element in the array.
- I is the array index.
- L is the starting index of an array.
- S is the size.

If we want to find the address of A [2] then it is calculated as: Loc (A [2]) =

Base + (I= L) * s

= 2000 + (2-0) * 2

= 2000 + 4

= 2004

Two Dimensional Array (Storage Representation)

Row-major Arrays: If two dimensional array elements store sequentially row byrow then it is called Row major array.

Example: A two dimensional array consist of two rows and three columns is storedsequentially in

RowRow	Column 0	Column 1	Column 2	
	A[0][0]	A[0][1]	A[0][2]	0
	A[1][0]	A[1][1]	A[1][2]	1

A[0][0]	
A[0][1]	ROW 1
A[0][2]	
A[1][0]	
A[1][1]	ROW 2
A[1][2]	

:

Row major order as:

- The address of element A[i, j] can be obtained by evaluating expression:

  Loc (A [i, j]) = Base + ((i – L1) * n + (j – L2))* s

  i is the row index.

  j is the column index.

  m is the no. of rows.

  n is the no of column.S is size.

  L1 is the lower bound for row.

  L2 is the lower bound for column.

- for example the address of element

A[1,2] in a[2][3] is calculated as:A [1, 2] = **Base + ((i− L1) * n + (j− L2))* s**

Here, n=3, m=2, i=1, j=2, Base = 3000

=3000 + ((1 - 0) * 3 + (2 - 0))*2

=3000 + (3+2)*2

=3000 + 10

=3010

Column-major Arrays: If two dimensional array elements store sequentiallycolumn by column then it is called Column major array.

- **Example:** A two dimensional array consist of two rows and three columns isstored sequentially in row major order as:

	Column 0	Column 1	Column 2
Row 0	A[0][0]	A[0][1]	A[0][2]
Row 1	A[1][0]	A[1][1]	A[1][2]

A[0][0]
A[1][0]
A[0][1]
A[1][1]
A[0][2]
A[1][2]

- The address of element A[i, j] can be obtained by evaluating expression:

  Loc (A [i, j]) = Base + ((j – L2) * m + (i – L1))* s

  i is the row index.

  j is the column index.m is the no.

  of rows.

  n is the no of column.S is size.

  L1 is the lower bound for row.

  L2 is the lower bound for column.

- for example the address of element A[1,2] in a[2][3] is calculated as:A [1, 2] = **= Base +**

  ((j= L2) * m + (i= L1))* s

  Here, n=3, m=2, i=1, j=2, Base = 3000

  =3000 + ((2 - 0) * 2 + (1 - 0))*2

  =3000 + (4+1)*2

  =3000 + 10

  =3010

Characteristics of Array.

- Array store elements that have same data type.
- Array store elements in subsequent memory location.
- Array size should be mention in the declaration.
- Array name represent the address of starting elements.
- Two dimensional array elements are stored row by row in subsequentmemory location.
- Insert and Delete operation are slower.
- Searching and sorting are faster.

List advantages of Array.

- We can use one name for similar objects and save them with same name butdifferent indexes.
- 2- D array are used to represent matrices.
- It can be used to implement other data structure like linked list, stack,queue ,tree, graph etc.

List Disadvantages of Array.

- We must know in advance how many elements are to be stored in array.
- If we allocate more memory than requirement than the memory space willbe wasted.

❖ 1.8 Overview of different array operations

- ✓ Insertion: Add a New data element into the list.
- ✓ Deletion: Remove an element from the list.
- ✓ Searching: Finding the specific location for specific data.
- ✓ Sorting: Arrange the list in Ascending or Descending order.
- ✓ Merging: Combine the two lists into single list.
- ✓ Traversal: Accessing each data element once in such a way that allthe data elements are processed.

❖ 1.9 Searching an element into an array:

Define Searching. Also write types of searching.

Searching: Search is used to find whether the desired data is present in set ofdata or not. If it is present, the search operation provides us its position.

There are two types of searching technique.

1) Linear search / Sequential Search
2) Binary Search

Explain Linear Search or Sequential Search (Explanation & algorithm)

Or

Write an algorithm for linear search. (Only algorithm)

- This method is used to find out element in an unordered list.
- Suppose the list L consist of N elements and we want to find the element fromthe list.
- It sequentially compares the given value with each value in list of values oneby one from start to end until value is found or list is ended.
- If the value is found, the position of the value is returned.

SEQENTIAL_SEARCH (L, N, X): These function searches the list L consist ofN elements for the value of X.

L: Represents the list of element.

N: Represents the no of elements in the list X: Represent the value to be search in the list

[Initialization]

i←0

Flag←1

1. [Search the List]
   Repeat thru step 3 for i=0, 1, N-1
2. [Comparison]
3. If (L[i]= X)

   Then (i) Flag ← 0

   (ii) Write(" Search is Successful")

   (iii) Write (" Value found at location: i +1")
4. if (flag=1) then

   Write ("Search is unsuccessful")
5. [Finished]

   Exit

Advantage:

(1) Very Simple

(2) Easy to understand and implement

(3) Works on both sorted as well as unsorted data, but mainly used for unsorteddata.

Disadvantage:

(1) Search Time is not unique, if value is in start it takes less time, but if value isnear to end it takes more time.

(2) As the size of data increase, the search time also linearly increase.

Explain Binary Search (Explanation & algorithm)

Or

Write an algorithm for Binary search. (Only algorithm)

- This method is used to find out element in an ordered list.
- It is very efficient method to search element.
- It first computes the middle which is index of the middle value.
- Then it compares the value to be search with middle element.
- If both matches, the value is found and index of middle element is returned.
- If both doesn't matches, then list is divided into two parts called lower half and upper half.
- All the value in lower half are less than the middle element and all the value in upper half are greater than the middle element.
- If value to be search is less than middle element then we have to repeat the process with the lower half and if value to be search is greater than middle element then we have to repeat the process with the greater half.

BINARY_SEARCH (L, N, X)

L: Represents the list of element.

N: Represents the no of elements in the list

X: Represent the value to be search in the list.

- LOW, HIGH and MIDDLE denotes the lower, upper and middle limit of the list.

[Initialize]LOW←0

HIGH←N-1

Flag←1

1. [Perform Search]

   Repeat thru step 4 while LOW ≤ HIGH

2. [Obtain Index of mid point interval]

3. MIDDLE ←[(LOW + HIGH)/2]

4. [Compare]

If X < L [MIDDLE] then

HIGH ← MIDDLE – 1

Else if X > L [MIDDLE] then

LOW←MIDDLE+1

Else

Write “Successful Search”

Flag←0

Return (MIDDLE)

5. If (Flag=1)

Write “Unsuccessful Search”

6. [Finished]

Exit.