Data Structure

Mr. Muhammad Yousaf MS (CS)
Lecturer
Department of Computer Science,
National University of Computer and Emerging Sciences, Pakistan.

Outline

- ADT’s
- Primitive Datatypes
- Non-Primitive Datatypes
- Linear Data Structures
- Non-Linear Data Structures

Introduction

- **DATA:** The quantities, characters or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical or mechanical recording media.

- You can think of a data as quantities, characters or symbols on which we can perform operations, we can store them, we can transmit them obviously in the form of electrical signals in our computer.

- Example C = a + b

Introduction

When data becomes information?

- Faisalabad belong I to
- reverse this whole string: I belong to Faisalabad
  - Now I can read this and I can understand what is written
- Meaning, if data is arranged in a systematic way, then it gets a structure and become meaningful.
- This meaningful or processed data is called information.
- We need to manage out data in such a way so that we can produce some meaningful information.

Introduction

- Data structure gives us the way to structure the data, to appropriately manage the data, so that we can get some meaningful information from it.
- We can use it whenever required in an efficient way.

Introduction

- **Data Structure:** Data structure is representation of the logical relationship existing between individual elements of data.

- In other words, a data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.

- A data structure is the systematic way to organize data so that it can be used efficiently.

Introduction

- We are arranging the data, we are organizing the data, we are managing the data in such a way so that we can use it efficiently.
- Efficiency means in terms of time as well as space.

Introduction

- Data structure affects the design of both structural & functional aspects of a program. Program=algorithm + Data Structure
- A algorithm is a step by step procedure to solve a particular Problem.

Introduction

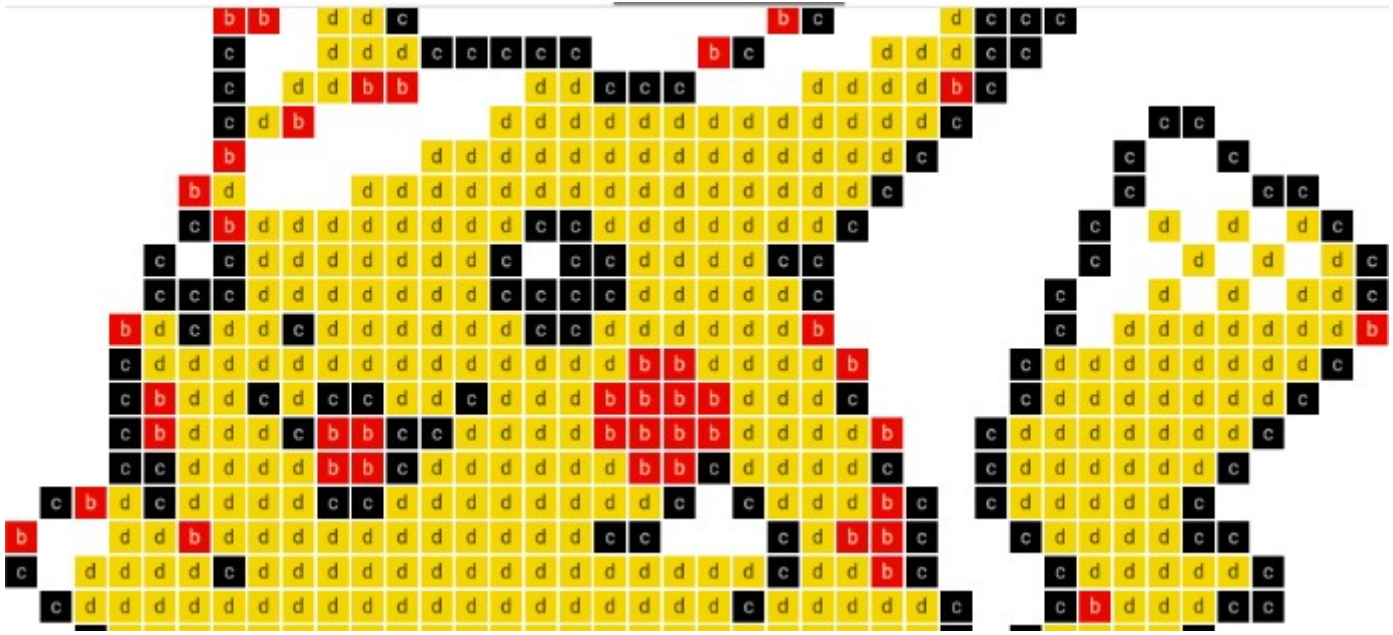
- Example (classical data structure, arrays).
  - We can think of it as a sequence or collection of some symbols, characters, integers, floats etc.
  - It is helping us in managing the data in an appropriate way, i.e., in a sequential manner.
  - Instead of creating multiple variables of same type, why not create an array to store all the values?
  - That is why, arrays are so useful.
- Example (Storing strings).
  - Strings are sequence of characters.
  - Let a string consist of 100 such characters. Now, are you going to create 100 variables for that?
  - No, you can create one array which can store that whole string.
  - And we would be able to access it very easily.
- Array helps us in managing our data in an appropriate way. That is why, it is a very useful data structure for us.

Introduction

- Redo undo: Stack data structure is used in implementing redo and undo feature

  - Google docs, or power point, or Microsoft word

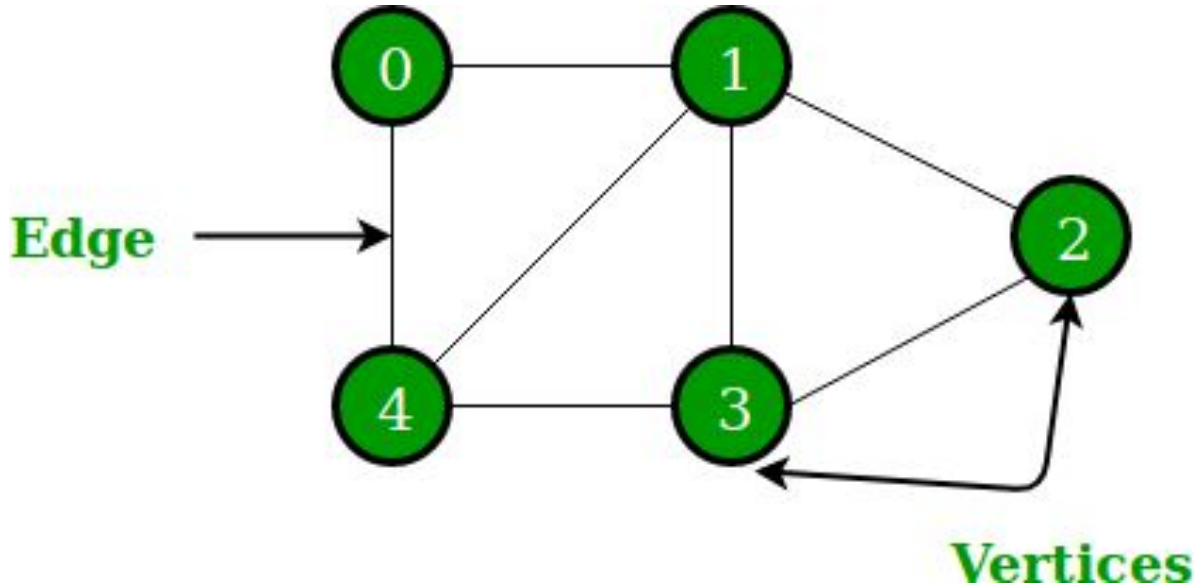
Introduction



- bitmap images
  - Array data structure is used to store that image
  - Array is so useful that, it is used in storing an image as well
  - Bitmap images are stored as a series of tiny dots called pixels
  - Each pixel is actually a small square that is assigned a color and then arranged in a pattern to form an image like the following.
  - So, you can think of a two dimensional array like this.

Introduction

- Storing the friendship information on a social networking site.
  - Graph Data Structure



Data Types

- It defines a certain domain of values.
- It defines operations allowed on those values.
- int type.
- Takes only integer values.
- Operations allowed are addition, subtraction, multiplication, bitwise operations et cetera.

Data Types

- Example: int type.
  - Takes only integer values.
  - Operations allowed are addition, subtraction, multiplication, bitwise operations et cetera.
- Example float type. T
  - Takes only floating point values.
  - Operations addition, subtraction, multiplication, division
  - Cannot perform bitwise and % mod operations on them.

User defined Data Types

- The operations and values of user defined data types are not specified in the language itself.
- It is specified by the user.
- The user is defining the operations as well as the values.
- It is not pre defined in the language itself.
- User define the values and operations

User defined Data Types

- Examples: Structure, union and enumeration
- By defining structures, we are defining our own type by combining other data types.
- By using structures, we are defining our own type, by combining other primitive types.

```
Struct point{
        Int x;
        Int y:
        }
```

- So, it's a user defined data type.

ADTs or Abstract data types

- They are like user defined data types which defines operations on values using functions without specifying what is there inside the function and how the operations are performed.

Stack ADT

- A stack consists of elements of same type arranged in a sequential order. (Allowed

  elements of same type + arranged in a sequential

- Operations
  - initialize()- initializes the stack to be empty actually.
  - Push() operation that is, insert an element into the stack.
  - Pop() operation that is, delete an element from the stack.
  - isEmpty(), check stack is empty.
  - isFull(), to check stack is full.
- Here, we are specifying function but we are not saying anything how they can be implemented. That is called abstract data type. We know that, what type of elements are allowed and we also know that what operations we can perform, but we don't know what is there inside.

ADTs or Abstract data types

- ADT can be think as a black box which hides the inner structure and design of the data type from the user.
- We can think of it like, it hides all the implementation details from us.
- There are multiple ways to implement an ADT.

ADTs or Abstract data types

- Example, a stack ADT can be implemented using arrays or linked lists.
- It should be well noted that stack itself is a data structure, we can implement this data structure using other data structures like arrays or linked lists.
- A stack ADT, which we know is right now is a skeleton, can be implemented using arrays or linked lists.

Why we use ADTs

- The program which uses data structure is called a client program.
  - It has access to the ADT that is the interface, nothing else.

    Just abstract.

- The program which implements the data structure is known as the implementation.
- Implementation is the one which implements the data structure and the client program is the one which just uses the interface, that is the outside details, nothing inside.

Why we use ADTs

- Example
  - If someone wants to use the stack in the program, then he can simply use push and pop operations without knowing its implementation.
  - A user doesn't have to worry about how the operations are performed.
  - If someone wants to use the stack program, he can simply use push and pop operations without knowing its implementation.

Why we use ADTs

- if in future, the implementation of stack is changed from array to linked list. The client program will work in the same way without being affected. There is nothing to discuss with client program.

- We are actually separating the two worlds. We are just providing the user, an interface. The rest of the details, the implementation part is done in the back end. User doesn't have to worry about it. The implementation can go on without affecting the client program.

Why we use ADTs

- Abstract data type provides abstraction.
- We should understand this, that abstract data type provides abstraction and it is very important that particular thing is which meanshidingdetails from theuser because user doesn't botherabout how implemented. He just have to use that thing.

Data Structure

- We know that a stack ADT can be implemented using arrays or linked lists.
- We can use these two data structures to implement a stack ADT. These are my data structures.
- A data structure is the organization of the data in a way so that it can be used efficiently. Efficiently, means in terms of time as well as space.
- A data structure is used to implement an ADT

Data Structure

- In order to implement stack ADT, we can use an array data structure or linked list data structure.
- We can use any of these data structures to implement this ADT. So, basically we can say that a data structure is used to implement an ADT
- ADT tells us what is to be done and data structures tells us how to do it.

Data Structure

- In order to implement stack ADT, we can use an array data structure or linked list data structure.

- We can use any of these data structures to implement this ADT. So, basically we can say that a data structure is used to implement an ADT

- ADT tells us what is to be done and data structures tells us how to do it.

- ADT just gives us the blue print, while data structures gives us the implementation

Data Structure

- How to know which data structure to use for a particular ADT?
- Stack ADT
  - Linked lists or Arrays
  - Different implementations of ADT are compared for time and space efficiency
  - We will select the one according to the current requirement of the user.

    May be user wants to save space, then we can use a particular data structure which can save space. May be a user wants run the program faster, then we will use the data structure, which will run faster.

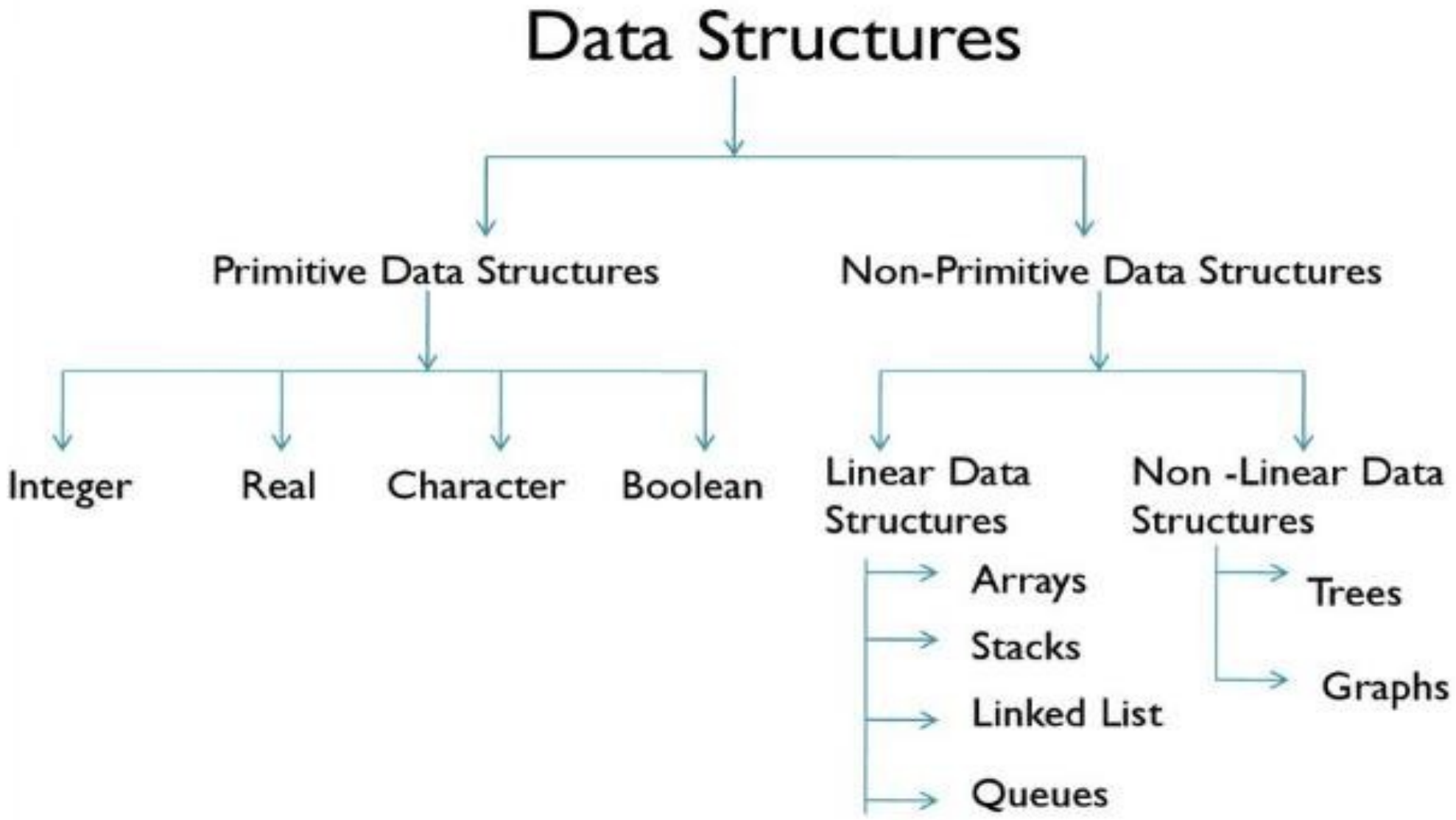
Advantages of Data Structure

- Efficiency. We know that, proper choice of data structure make program efficient in terms of space and time.
- Reusability. One implementation can be used by multiple client programs.
- Abstraction. Data structure is specified by an ADT. ADT is the blueprint, while data structure is the implementation. We can say that, it provides a level of abstraction. The client program doesn't have to worry about what is going inside. It just sees the ADT, just the interface, nothing else.

Classification of Data Structure

- Data structure are normally divided into two broad categories:
  - Primitive Data Structure
  - Non-Primitive Data Structure.

Classification of Data Structure



Primitive vs Non Primitive Data Structures

- Primitive data structure :-The data structures, that are directly operated upon by machine level instructions i.e. the fundamental data types such as int, float in case of ‘c’ are known as primitive data structures.

- Non-Primitive data structure :-These are more complex data structures.
  - These data structures are derived from the primitive data structures.
  - The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.
  - Non-Primitive Data Structure Lists, Stack, Queue, Tree, Graph are example of non-primitive data structures.

Non-Primitive Data Structures

Non-Primitive Data Structures can be further divided into two categories: Linear Data Structures and Non-Linear Data Structures.

- Linear Data Structures:- In linear data structures, data elements are organized sequentially and therefore they are easy to implement in the computer's memory. E.g. Arrays, stacks, linked lists and Queues.
  - Elements has only one predecessor and one successor except first and last as in array.
- Non-Linear Data Structures:- In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them. E.g. Graphs

Non-Primitive Data Structures

Non-Primitive Data Structures can be further divided into two categories:

Linear Data Structures and Non-Linear Data Structures.

- Non-Linear Data Structures:- In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them. E.g. Graphs
- A data structure is non linear when all the elements are not arranged in a linear or sequential order. There is no linear arrangement of the elements.

Non-Primitive Data Structures

The most commonly used operation on data structure are broadly categorized into following types:

- Create
- Selection
- Updating
- Searching
- Sorting
- Merging
- Destroy or Delete

Non-Primitive Data Structures

The most commonly used operation on data structure are broadly categorized into following types:

- Create
- Selection
- Updating
- Searching
- Sorting
- Merging
- Destroy or Delete

Primitive Vs Non-Primitive Data Structures

Roughly we can say that:

- A primitive data structure is generally a basic structure that is usually built into the language, such as an integer, a float.
- A non-primitive data structure is built out of primitive data structures linked together in meaningful ways, such as a or a

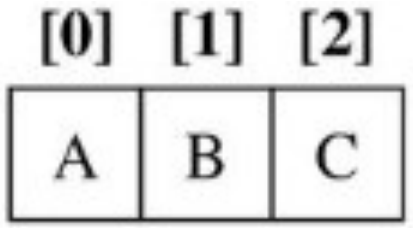
  linked-list, binary search tree, AVL Tree, graph etc.

Description of various Data Structures:

Arrays

- An array is defined as a set of finite number of homogeneous elements or same data items.
- It means an array can contain one type of data only, either all integer, all float-point number or all character.

Array



Description of various Data Structures:

Arrays

- Declaration of array is as follows: int arr[10]
  - Where intspecifies the data type or type of elements arrays stores.
  - “arr” is the name of array & the number specified
  - inside the square bracketsis the number of elements an array can store, this is also called sized or length of array.

Description of various Data Structures:

Arrays

- Following are some of the concepts to be remembered
  - The individual element of an array can be accessed by specifying name of the array, following by index or subscript inside square brackets.
  - The first element of the array has index zero[0]. It means the first element and last element will be specified as:
    - arr[0] & arr[9] Respectively.

Description of various Data Structures:

Arrays

- The elements of array will always be stored in the consecutive (continues) memory location.
- The number of elements that can be stored in an array, that is the size of array or its length is given by the following equation: (Upperbound - lowerbound)+1

Description of various Data Structures:

Arrays

- For the above array it would be (9-0)+1=10,where 0 is the lower bound of array and 9 is the upper bound of array.
- Array can always be read or written through loop. If we read a one- dimensional array it require one loop for reading and other for writing the array

Description of various Data Structures:

Arrays

- Reading an array

```
For(i=0;i<=9;i++)
        scanf("%d",&arr[i]);
```

- Writing an array

```
For(i=0;i<=9;i++)

        printf("%d",arr[i]);
```

Description of various Data Structures:

Arrays

- If we are reading or writing two- dimensional array it would require two loops.
- And similarly the array of a N dimension would required N loops.

Description of various Data Structures:

Arrays

- Some common operation performed on array are:
  - Creation of an array
  - Traversing an array
  - Insertion of new element
  - Deletion of required element
  - Modification of an element
  - Merging of arrays

Description of various Data Structures:

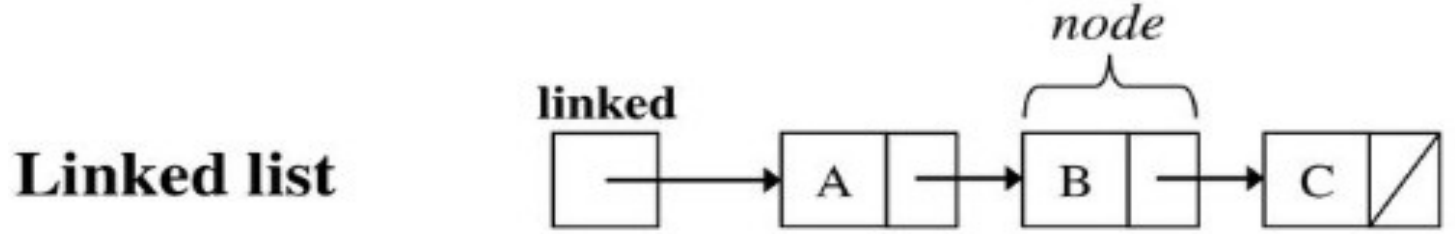
Arrays

- Some common operation performed on array are:
  - Creation of an array
  - Traversing an array
  - Insertion of new element
  - Deletion of required element
  - Modification of an element
  - Merging of arrays

Description of various Data Structures: List

- A lists (Linear linked list) can be defined as a collection of variable number of data items.

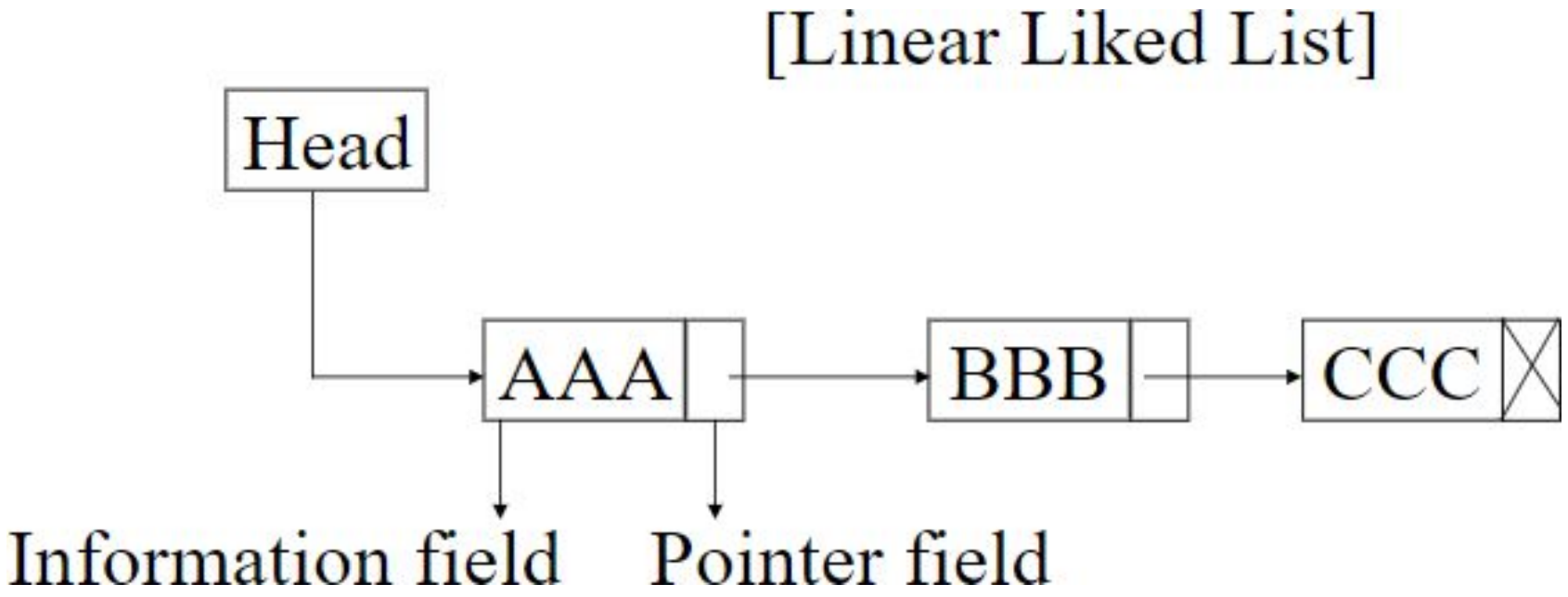
- Lists are the most commonly used non- primitive data structures.
- An element of list must contain at least two fields, one for storing data or information and other for storing address of next element.
- As you know for storing address we have a special data structure of list i.e., the address must be pointer type.



Linked list

Description of various Data Structures: List

- Technically each such element is referred to as a node, therefore a list can be defined as a collection of nodes as show bellow:



Description of various Data Structures: List

- Types of linked lists:
  - Single linked list
  - Doubly linked list
  - Single circular linked list
  - Doubly circular linked list

Description of various Data Structures: Stack

- A stack is also an ordered collection of elements like arrays, but it has a special feature that deletion and insertion of elements can be done only from one end called the top of the stack (TOP)
- Due to this property it is also called as last in first out type of data structure (LIFO).

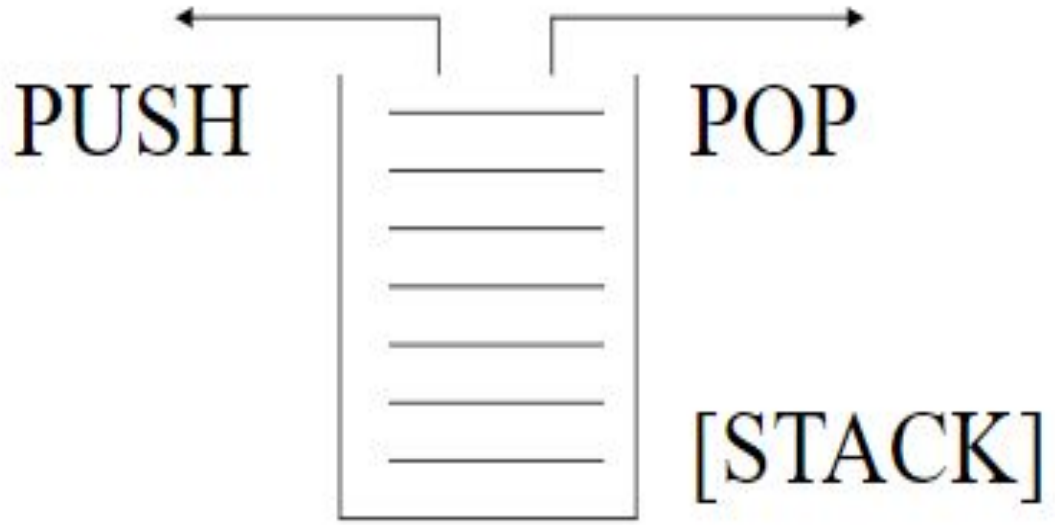
Description of various Data Structures: Stack

- It is just like a stack of plates placed on table in a party, a guest always takes off a fresh plate from the top and the new plates are placed on to the stack at the top.

- It is a non-primitive data structure.
- When an element is inserted into a stack or removed from the stack, its base remains fixed where the top of stack changes.

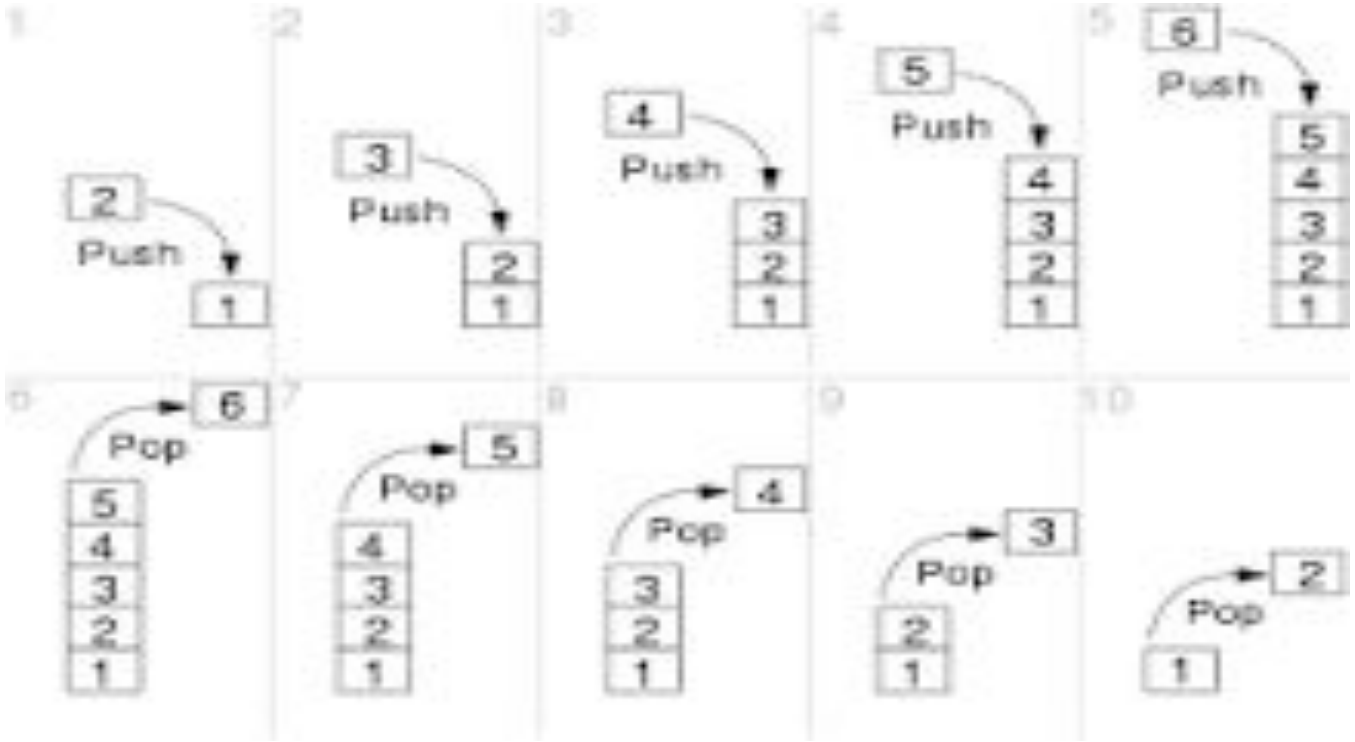
Description of various Data Structures: Stack

- Stack Insertion of element into stack is called PUSH and deletion of element from stack is called POP.
- The bellow show figure how the operations take place on a stack: PUSHPOP [STACK]



Description of various Data Structures: Stack

- A push operation decrements the pointer and copies the data to the stack; a pop operation copies data from the stack and then increments the pointer.



Description of various Data Structures: Stack

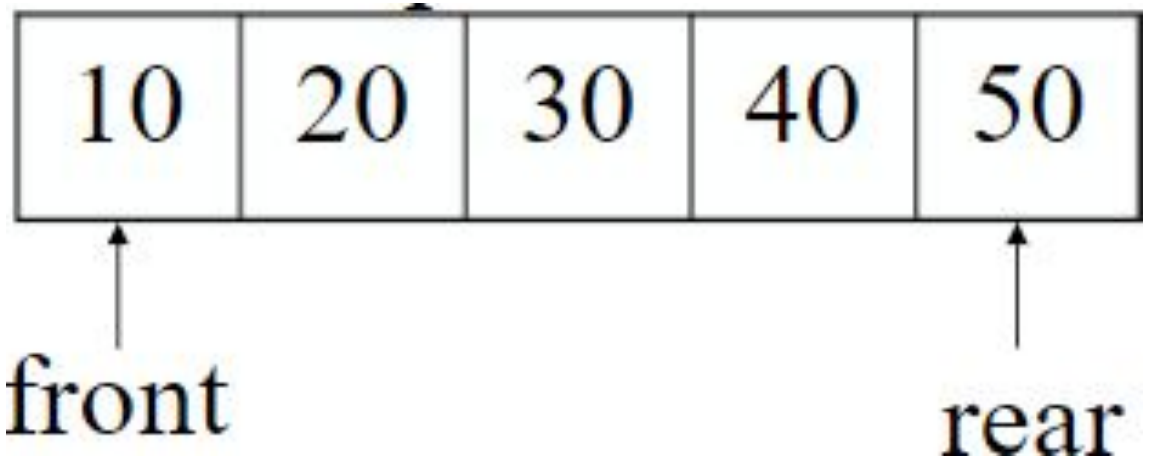
- The stack can be implemented into two ways:
  - Using arrays (Static implementation)
  - Using pointer (Dynamic implementation)

Description of various Data Structures: Queue

- Queue are first in first out type of data structure (i.e. FIFO)
- In a queue new elements are added to the queue from one end called REAR end and the element are always removed from other end called the FRONT end.
- The people standing in a railway reservationrow are an example of queue.

Description of various Data Structures: Queue

- Each new person comes and stands at the end of the row and person getting their reservation confirmed get out of the row from the front end.
- The bellow show figure how the operations take place on a stack:



Description of various Data Structures: Queue

- The queue can be implemented into two ways:
  - Using arrays (Static implementation)
  - Using pointer (Dynamic implementation)

Description of various Data Structures:

Tree

- A tree can be defined as finite set of data items (nodes).
- Tree is non-linear type of data structure in which data items are arranged or stored in a sorted sequence.
- Tree represent the hierarchical relationship between various elements.

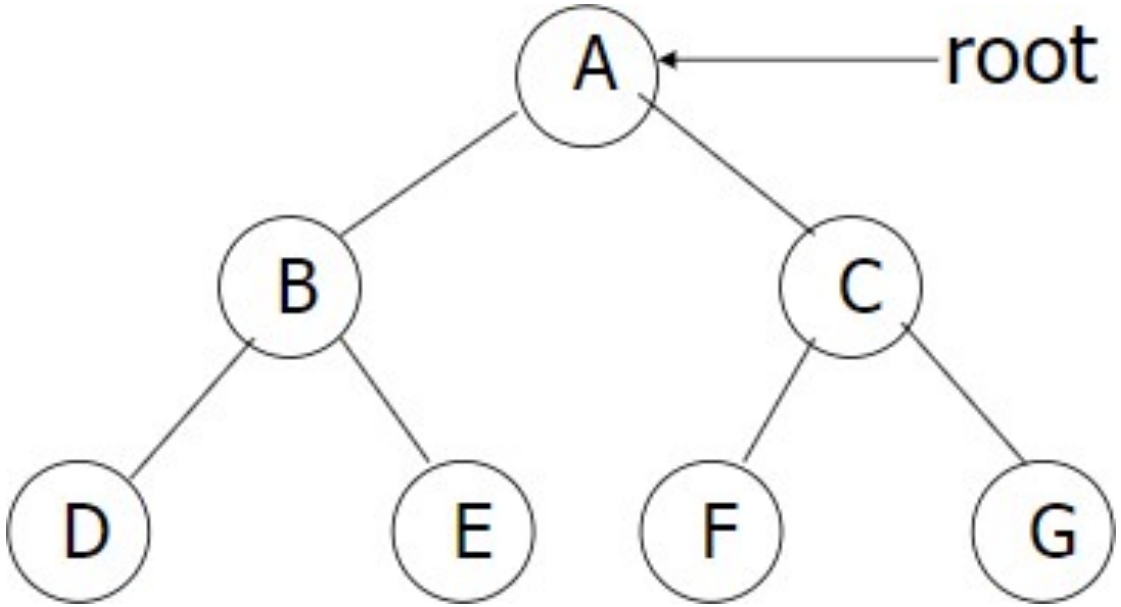
Description of various Data Structures:

Tree

- In trees: There is a special data item at the top of hierarchy called the Root of the tree.
- The remaining data items are partitioned into number of mutually exclusive subset, each of which is itself, a tree which is called the sub tree.
- The tree always grows in length towards bottom in data structures, unlike natural trees which grows upwards.

Description of various Data Structures: Queue

- The tree structure organizes the data into branches, which related the information.



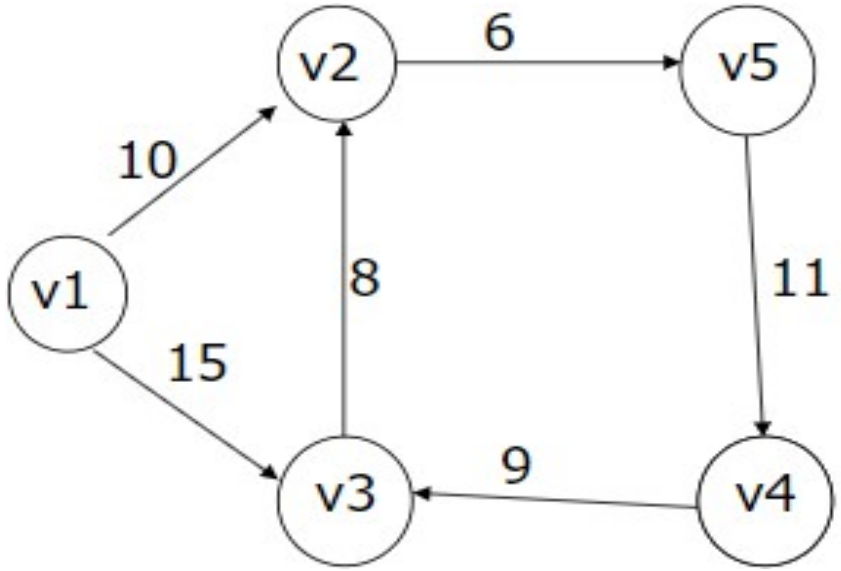
Description of various Data Structures: Graph

- Graph is a mathematical non-linear data structure capable of representing many kind of physical structures.
- It has found applicationin Geography, Chemistry and Engineering sciences.
- Definition: A graph G(V, E) is a set of vertices V and a set of edges E.

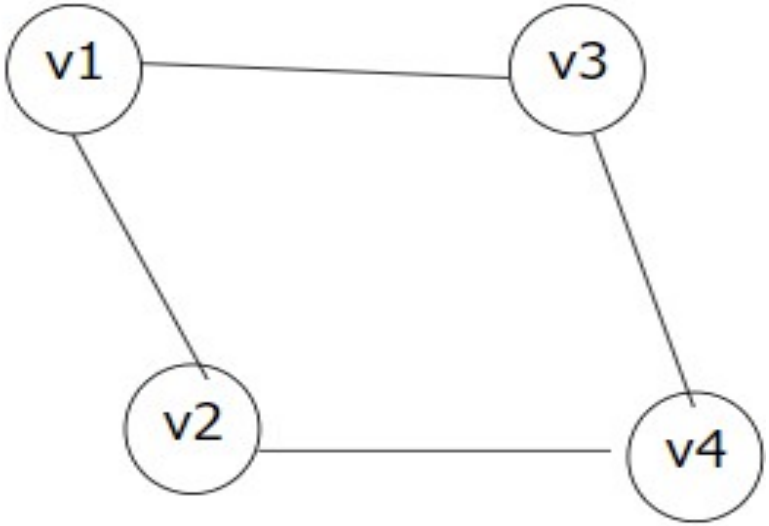
Description of various Data Structures: Graph

- An edge connects a pair of vertices and many have weight such as length, cost and another measuring instrument, according the graph.
- Vertices on the graph are shown as point or circles and edges are drawn as arcs or line segment

Description of various Data Structures: Graph



[a] Directed & Weighted Graph



[b] Undirected Graph

Description of various Data Structures: Graph

- Types of Graphs:

- Directed graph
- Undirected graph
- Simple graph
- Weighted graph
- Connected graph
- Non-connected graph

Static data structures and Dynamic data structures

- **Static Data Structure:**
  - In these type of data structure, the memory is allocated at compile time.
  - Therefore, maximum size is fixed.
  - The advantage of such data structure is that, we can access data very fast.
  - Disadvantage is that insertion and deletion operations are slower.
  - Array data structure is a static data structure, here you can see that, we have to fix the size beforehand.

Static data structures and Dynamic data structures

- Dynamic Data Structure:
  - In these type of data structures, the memory is allocated at run time.
  - Therefore, the maximum size is flexible.
  - When the memory will be allocated at run time, then user will decide the size.
  - In that case, the size is flexible.
  - Advantage is of course, faster insertion and deletion.
  - Disadvantage is slower access.
  - Example Linked list

Thank
You