Data structures and Algorithms

General information

CSC 2100	: Data Structure and Algorithm
Instructor	: David Byansi
Phone	: 0785 901241/ 0757 887502.
Email	: byansid4@gmail.com/byansi.david@cis.mak.ac.ug
Lecture time	: Thursday 8:00am – 10:00 am. (LLT3B) Friday 11:00am – 1:00 pm. (LLT3B)
Office hours	: By appointment

Course Objective

- Understand the concept of data structure
- Distinguish between data structures and data types
- Learn how data structures are organized in the computer
- Describe the characteristics of common types of data structures
- Understand the concept of Abstract Data types
- Describe the advantages of Abstract Data Types
- Understand the concept of algorithm
- Describe the properties of algorithm
- Discuss the implementation of Algorithms

Motivating Quotations

- "Every program depends on algorithms and data structures, but few programs depend on the invention of brand new ones."

  --- Kerninhan & Pike

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationship."

- - Linus Torvalds

Introduction

- Around the globe and because of technological advancement, it is difficult to spot a discipline that has distanced itself from the discipline of computer science.
- To quote a few,
  - Medical surgery or diagnosis performed by robots or doctors on patients half way across the globe
  - Launching of space crafts and satellites into outer space
  - Forecasting tornadoes and cyclones
  - Online reservation of tickets
  - Billing at food store
  - Control of washing machines.
- One need computers to be omnipresent, omnipotent

Introduction

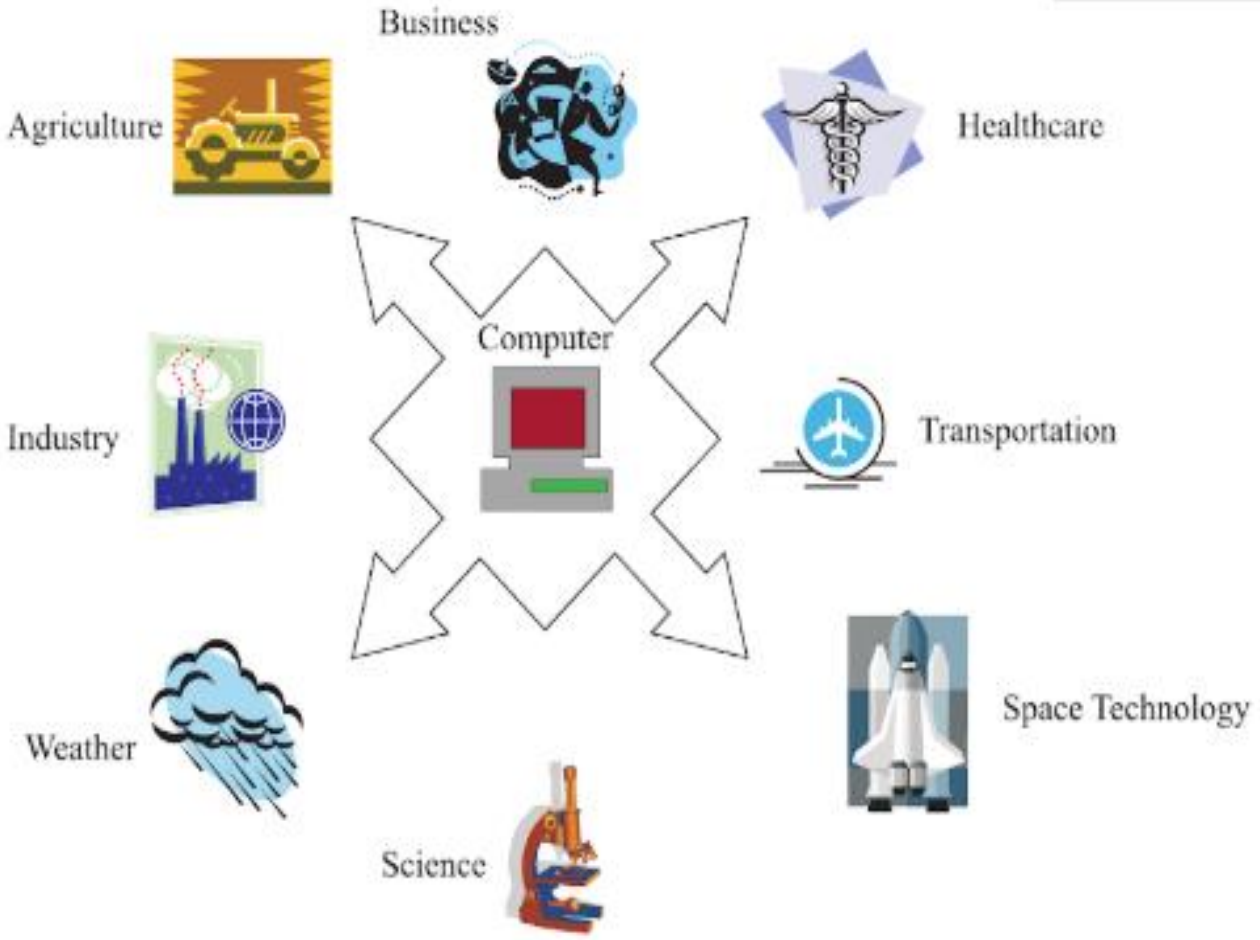


Fig. 1.1 *Omnipresence of computers*

Introduction

- In simple terms, any discipline that calls for problem-solving using computers , look up to the discipline of computer science for efficient and effective methods and techniques of solutions to the problems in their respective fields.
- To remind ourselves
  - **Computer science** is simply the study of problems, problem-solving and the solution come out of the problem-solving process through the design of algorithms.

Introduction

- From the point of view of problem-solving, the discipline of computer science could be categorized into the following four sub areas.
  - **Machines –**
    - What machines are appropriate or available for the solution of a problem?
    - What is the machine configuration- its processing power, memory capacity etc. that would be required for the efficient execution of the solutions?
  - **Languages**
    - What is the language or software with which the solution of the problem needs to be coded? What are the software constraints that would hamper the efficient implementation of the solution?
  - **Foundations**
    - What is the model of a problem and its solution? What method need to be employed for the efficient design and implementation of the problem? What is its performance measure?
  - **Technologies**
    - What are the technologies that need to be incorporated for the solution of the problem? For example, does the solution call for a web based implementation or needs activation from mobile devices or calls for hand shaking broadcasting devices or merely needs to interact with high end or low end peripheral devices.

Introduction

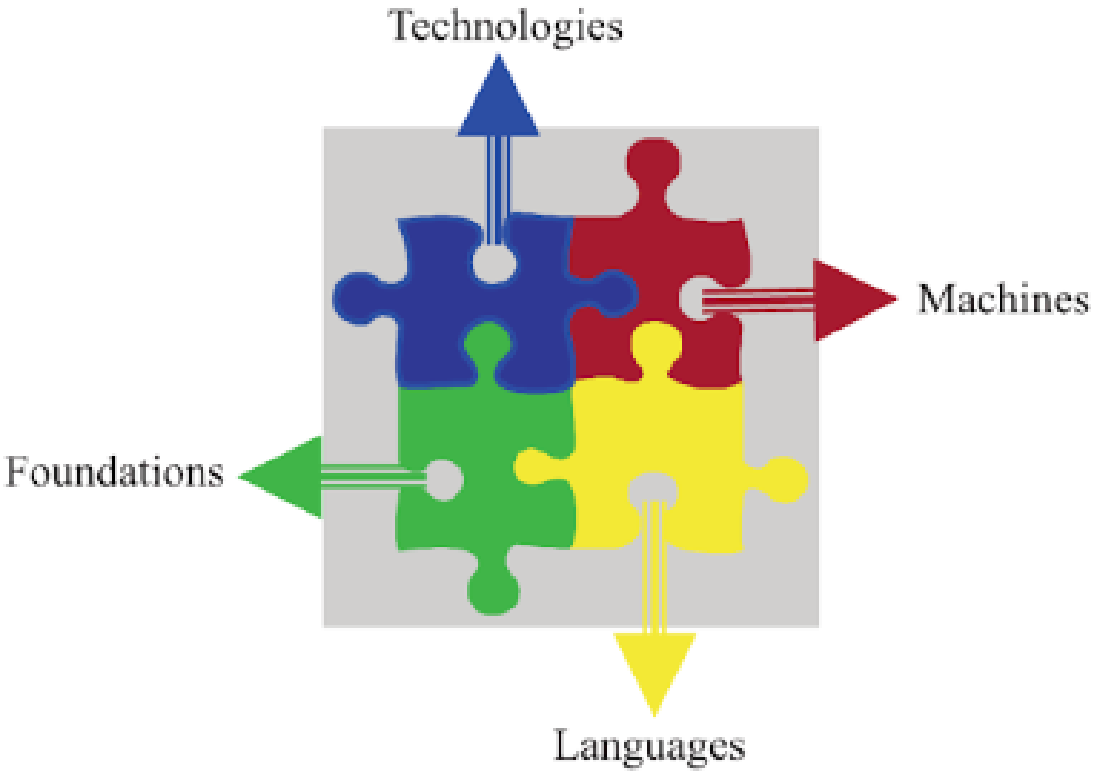


Fig. 1.2 *Discipline of computer science from the point of view of problem solving*

Introduction

- One of the core fields that belong to the foundations of computer science deals with the design, analysis and implementation of algorithms for efficient solution of the problems concerned.
- An **Algorithm** may be loosely defined as a process or procedure or method or recipe. It is a specific set of rules to obtain a definite output from specific inputs provided to the problem.
- **Data structures** is connected with the design and implementation of efficient algorithms.
- Data structures deals with the study of methods, techniques and tools to organize or structure data.

Definition, Structure and Properties of Algorithms

- **Definition**
  - An **algorithm** may be defined as a finite sequence of instructions each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time.
- **Structure and Properties**
  i. Input step
  ii. Assignment step
  iii. Decision step
  iv. Repetitive step
  v. Output step

Example of structure and properties of an algorithm

- Consider an example of adding two numbers 987 and 76.
  - In this, the input step considers the two operands 987 and 76 for addition.
  - The assignment step sets the pair of digits from the two numbers and the previous carry digit if it exists, for addition.
  - The decision step decides at each step whether the added digits yield a value that is greater than 10 and if so, to generate the appropriate carry digit.
  - The repetitive step repeats the process for every pair of digits beginning from the least significant digit onwards.
  - The output step releases the output which is 1063

Properties/Characteristics of algorithms

- An algorithm is endowed with the following properties:
  - **Finiteness-**an algorithm must terminate after a finite number of steps.
  - **Definiteness-** the steps of the algorithm must be precisely defined or unambiguously specified.
  - **Generality-**an algorithm must be generic enough to solve all problems of a particular class
  - **Effectiveness –** the operations of the algorithm must be basic enough to be put down on pencil and paper. They should not be too complex to warrant writing another algorithm for the operation
  - **Input-Output –** the algorithm must have certain initial and precise inputs and outputs that may be generated both at its intermediate and final steps.

Properties/Characteristics of algorithms

- An algorithm does not enforce a language or mode for its expression but only demands adherence to its properties.
- Thus one can write an algorithm in one's own expressive way to make a cup of hot coffee.
  - However, there is this observation that a cooking recipe that calls for instructions such as "add a pinch of salt and pepper', 'fry until it turns golden brown' are "anti-algorithmic" for reason that terms such as 'a pinch', ' golden brown' are subject to ambiguity and hence violate the property of definiteness".
  - An algorithm may be represented using pictorial representations such as flow charts.
  - An algorithm encoded in a programming language for implementation on a computer is called **program**

Development of an Algorithm

- The steps involved in the development of an algorithm are as follows:
  i. Problem statement
  ii. Model formulation
  iii. Algorithm design
  iv. Algorithm correctness
  v. Implementation
  vi. Algorithm analysis
  vii. Program testing
  viii. documentation
- Once a clear state of the problem is done, the model for the solution of the problem is to be formulated.
- The next step is to design the algorithm based on the solution model that is formulated. It is here that one sees the role of **data structures**. The right choice of the data structure needs to be made at the design stage itself since data structure influence the efficiency of the algorithms.
- Once the correctness of the algorithm is checked and the algorithm implemented, the most important step of measuring the performance of the algorithm is done. This is termed as **algorithm analysis**.
- It can be seen how the use of appropriate data structures results in a better performance of the algorithm. Finally the program is tested and the development ends with proper documentation

- The design of an efficient algorithm for the solution of the problem calls for the inclusion of appropriate data structures.
- A clear, unambiguous set of instructions following the properties of the algorithm alone does not contribute to the efficiency of the solution.
- It is essential that the data on which the program need to work on are appropriately structured to suit the needs of the problem, thereby contributing to the efficiency of the solution

- For example, let us consider the problem of searching for a telephone number of a person, in the telephone directory.
  - This will be easy if the data is sorted according to alphabetical order of the subscriber's names
  - However when the data is unstructured, the search algorithm turns out to be crude and hence inefficient.
- This is a classic example to illustrate the significant role played by data structures in the efficiency of the algorithms.

Data Structures and Algorithms

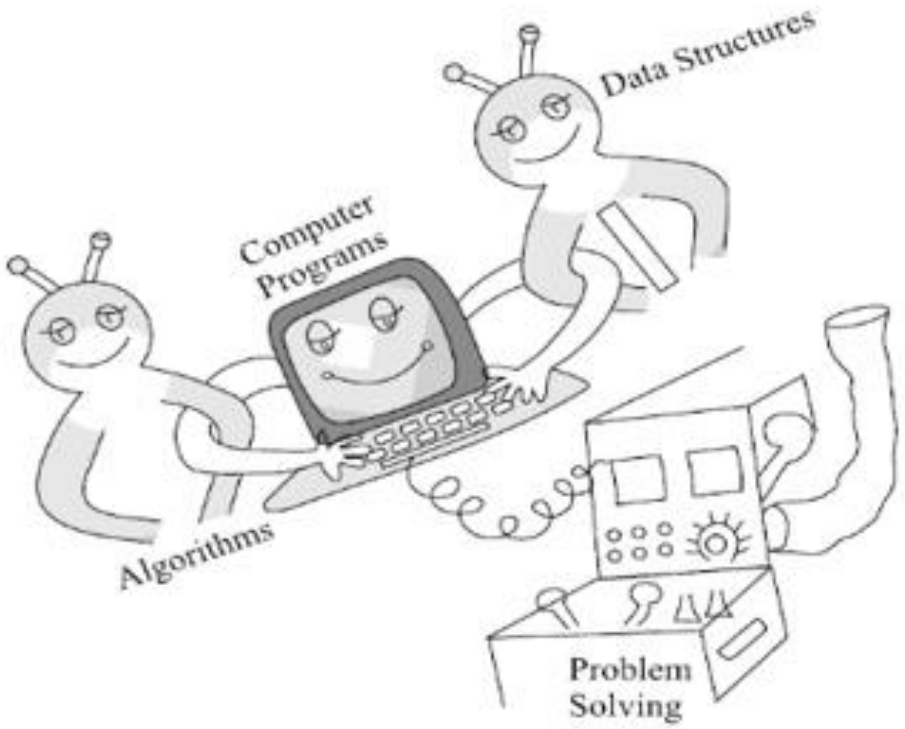
- All in all **data structures** is simply the study of various ways of organizing data in a computer so that the information can be manipulated by systematic and step by step procedure called **algorithms.**

- The data structure can be defined as the collection of elements and all the possible operations which are required for those set of elements. In other words data structure will tell us which are the elements required as well as the legal operations on those set of elements.

- **For Example**
  - Consider a set of elements which are required to store in an array. Various operations such as **reading** of the elements and **storing** them at appropriate index can be performed. If we want to access any particular element then that element can be retrieved from the array. Thus **reading, printing, searching** would be the operations required to perform these tasks for the elements. Thus integer elements and set of operations form the data structure-array

Data Structures and Algorithms

- Data structures and algorithms cannot work in isolation. They have to work hand in hand to give an ideal program which is able to solve a problem.
- **Algorithm + Data structure = Program**



Data Structure—Definition and Classification

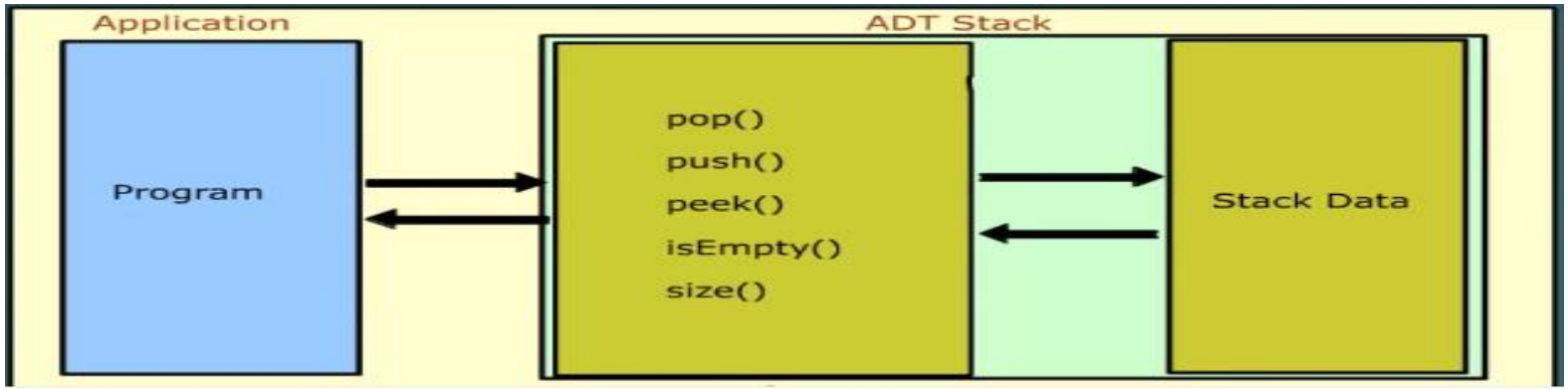
- Abstract data types
  - A **data type-** refers to the type of values that variables in a programming language hold. Thus data types of integer, real, character, Boolean which are inherently provided in programming languages are referred as **primitive data types.**
  - A list of elements is called a **data object**. For example a list of integers or a list of alphabetical strings as data objects
  - The data objects which comprise the data structure and their fundamental operations are known as **Abstract Data Type (ADT)**
  - **In other** words, an ADT is defined as a set of data objects D defined over a domain L and supporting a list of operations O.

Data Structure—Definition and Classification

- ADT has the following characteristics:
  - Provides a description of elements in terms of data types
  - Defines relationship among individual elements
  - Valid operations and parameters to be passed
  - Error conditions associated with the operations
- For example an ADT stack can be defined by packaging together the following operations
  - Push – insert an item into the stack
  - Pop – Returns a data item from stack
  - peek()- Returns top most element without removing it from the stack
  - size() – Returns the number of elements currently in the stack
  - isEmpty() – Returns true if stack is empty, and false otherwise.
- Set of operations associated with ADT is called **interface**.
- Elements of ADT data structure are manipulated through an interface

Data Structure—Definition and Classification

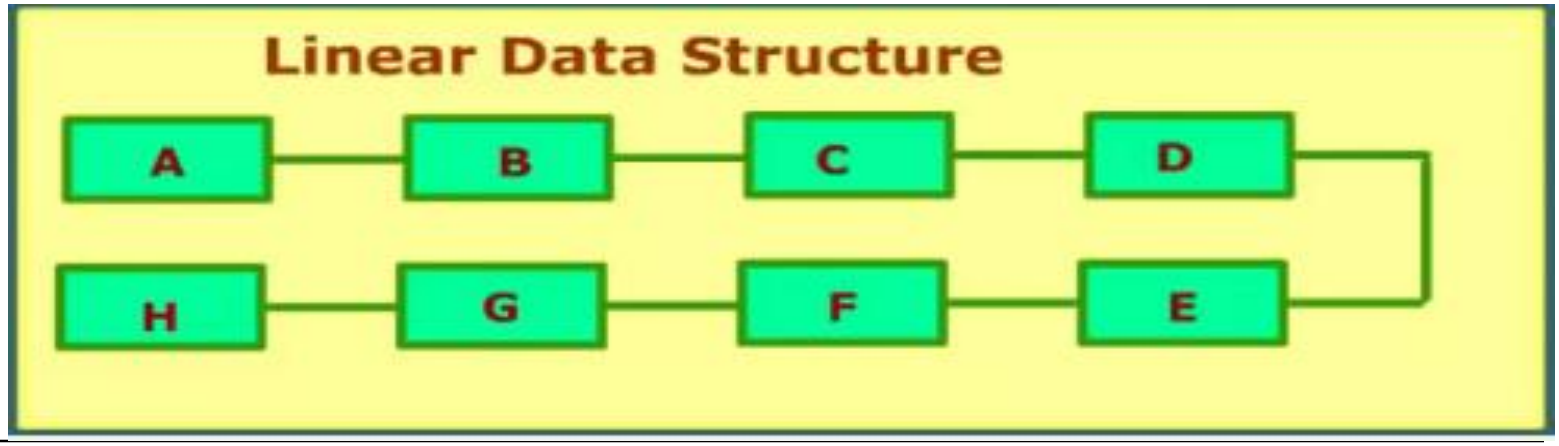
- The figure shows how elements of ADT are accessed using interface.
- Implementation of operations and data items are hidden from the application program.



Classification of data structures

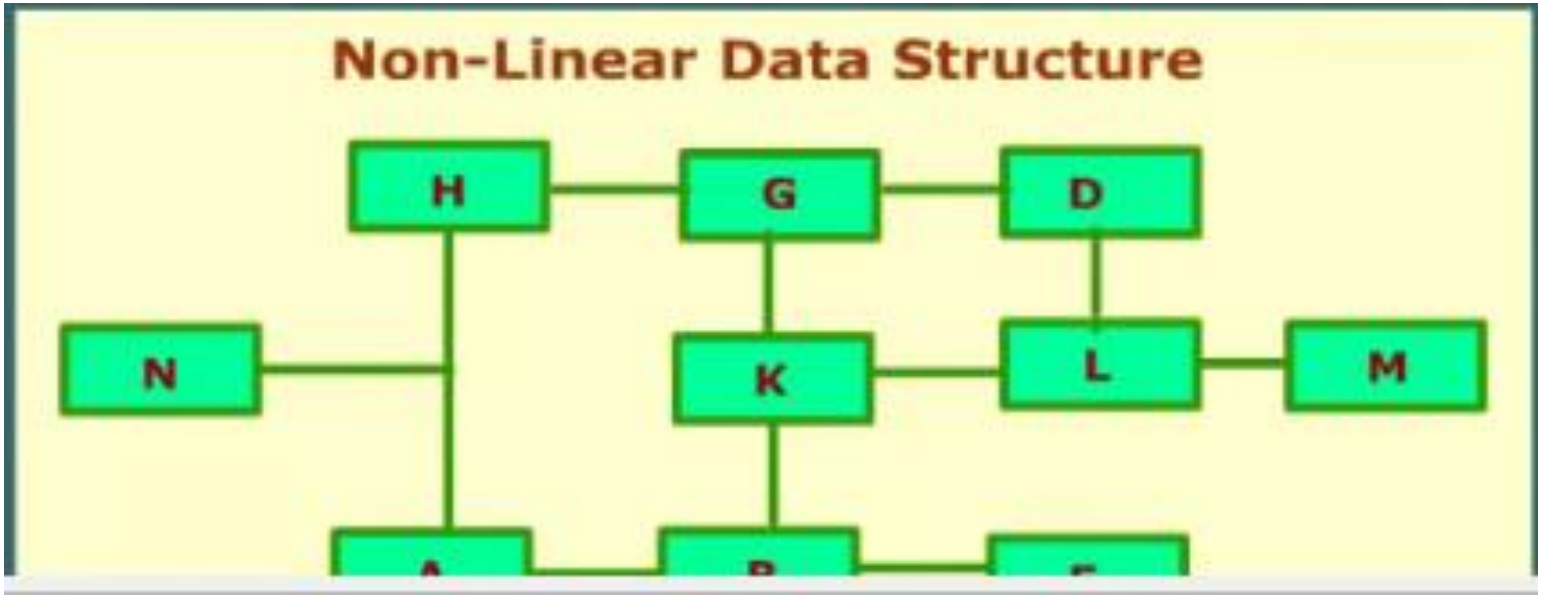
- The data structures are broadly classified as ***linear data structures*** and ***non-linear data structures.***

  - **Linear Data structures** –are data structures in which data is arranged in a list or in a straight line. They are further classified as sequential and linked representations.

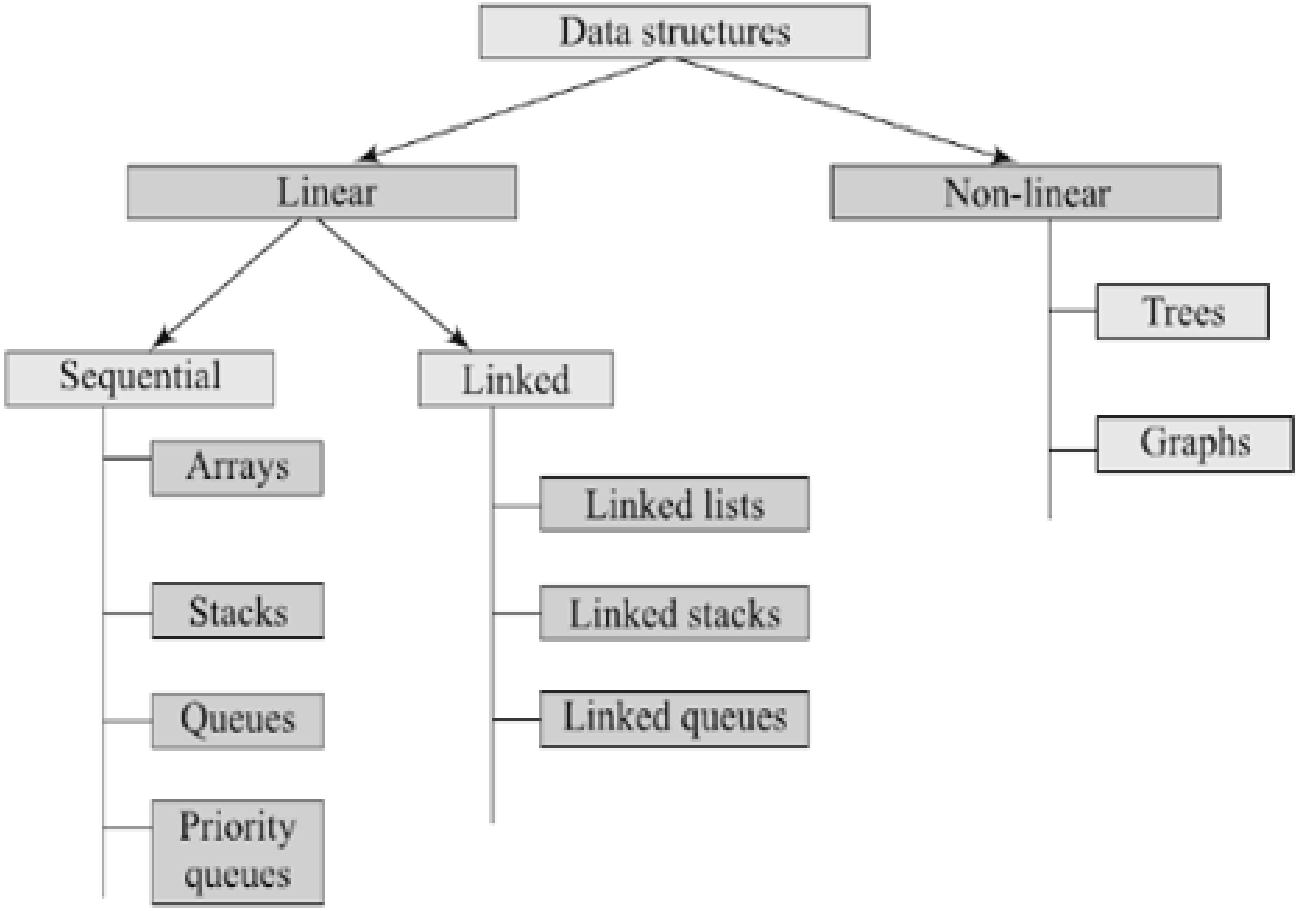


Classification of data structures

- **Non-linear data structures** are the data structures in which data may be arranged in hierarchical manner.

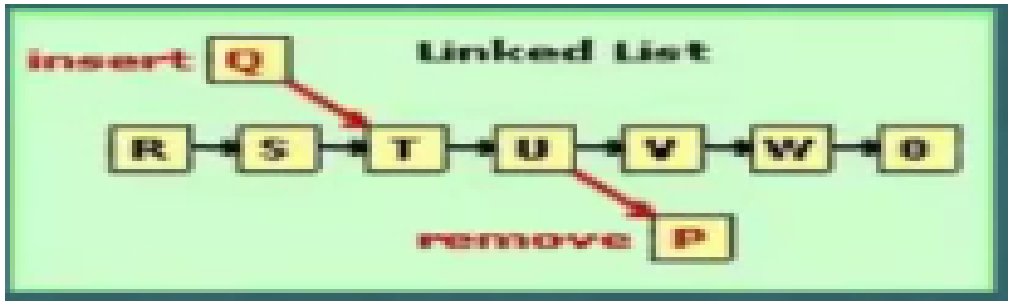


Classification of data structures

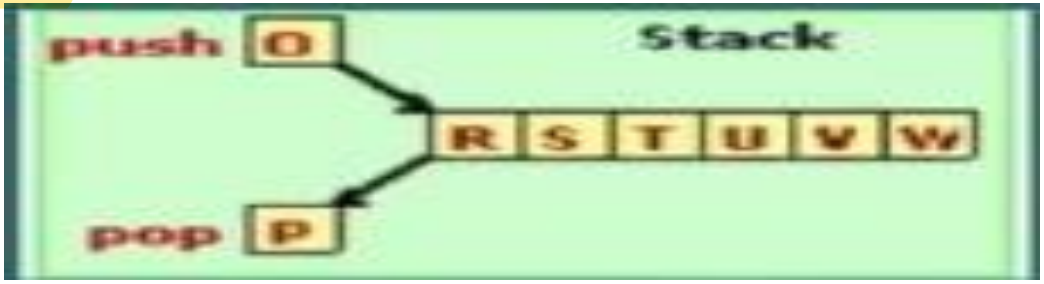


Classification of data structures (Linear)

- **Linked list** consist of data items called nodes. Each data element contains two field; information field and a link field connected to another data items.

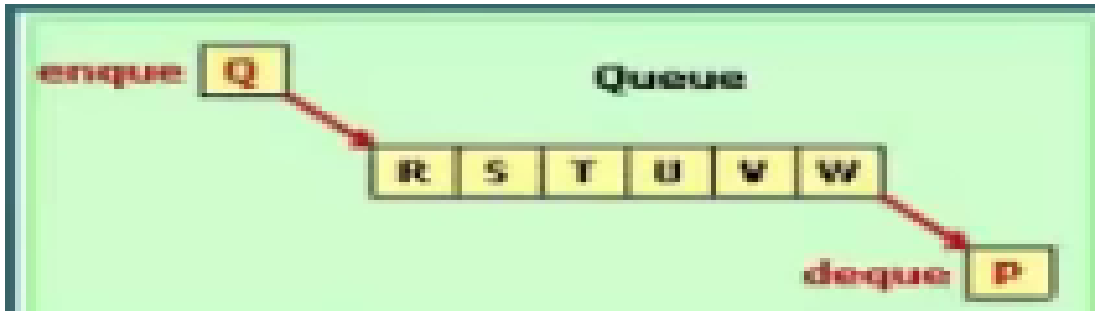


- In **a stack**, last item added can be taken off first. Works on the principle of LIFO

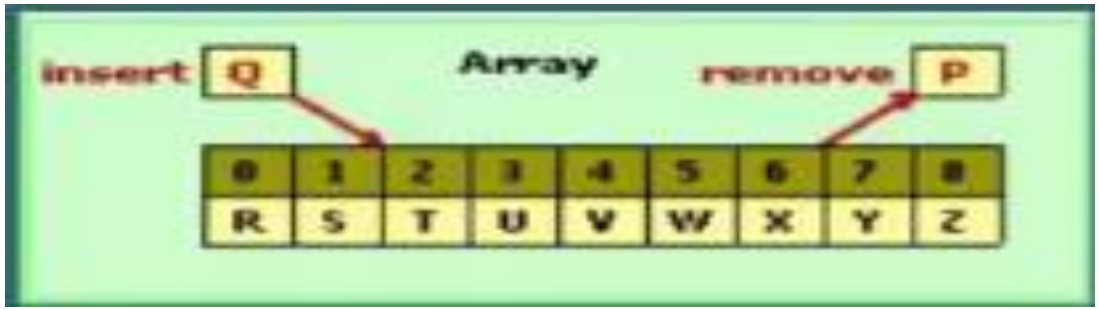


Classification of data structures (Linear)

- A queue is a data structure in which an item can be added at one end and removed from the other end. It works on the principle of FIFO

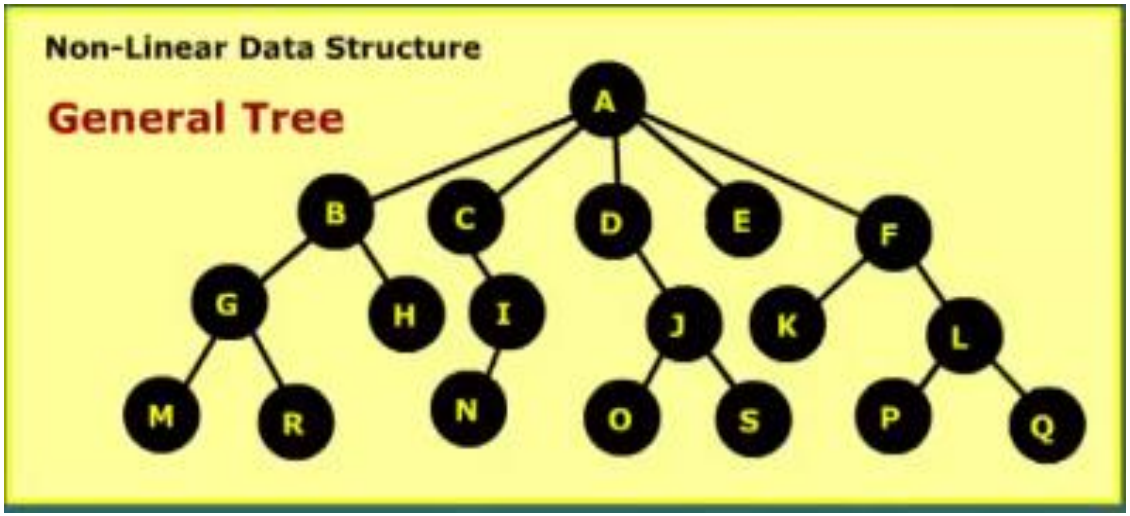


- An array is a collection of homogeneous items.



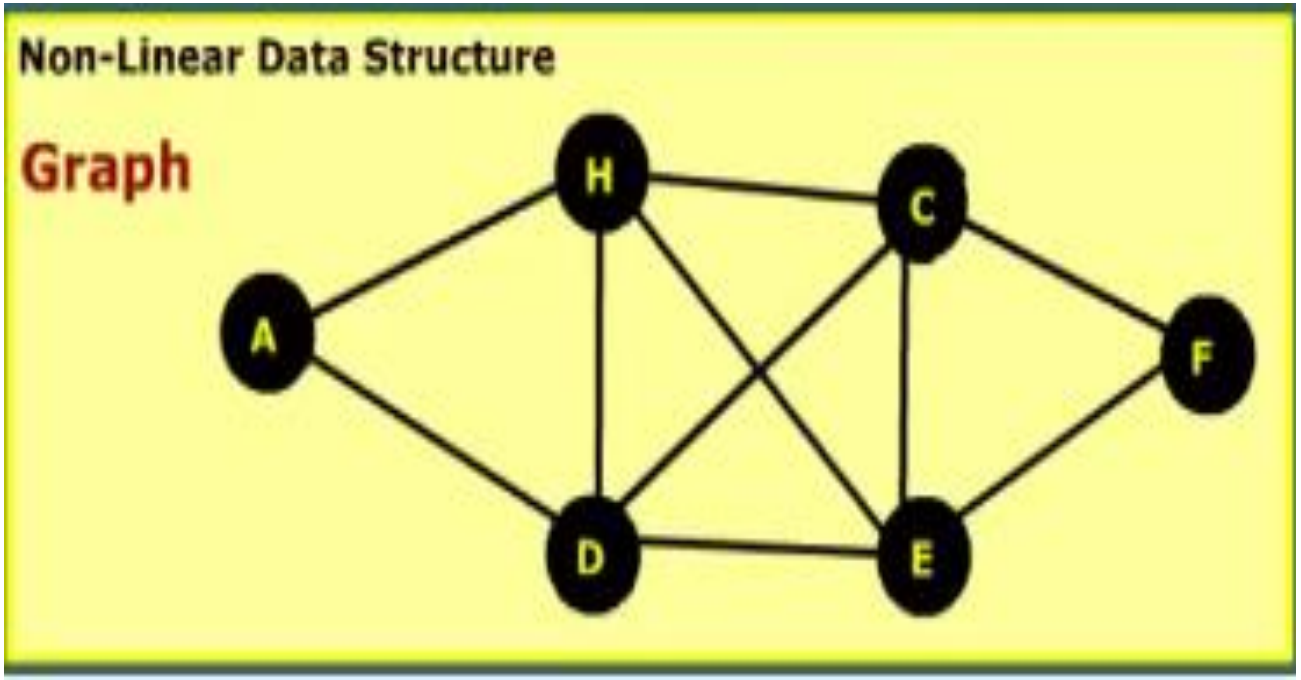
Classification of data structures (Non-Linear)

- A **tree** is a data structure that consists of a set of data items called nodes.
  - It contains an information field and a link filed, which is address of related element
  - There are variations of trees i.e binary, AVL, Red and Black, Splay, B-trees or heap



Classification of data structures (Non-Linear)

- **Graph** is a set of data items, called vertices or nodes, connected by links termed as edges or arcs.



Operations on Data Structures

- In order to process elements of a data structure some kind of operation is required.
- A basic operation involves **creating** a data structure and reserving storage space for the elements.
- Another operation is **deleting** the data structure, which removes the data items and releases the allocated storage space.

Operations on Data Structures

- Other common operations are: inserting, sorting, searching, merging, traversing.
  - **Inserting** involves adding a data item in a specified position to the data structure.
  - **sorting** is arranging of data items in specific order.
  - **Searching** means finding an item which matches with a given key.
  - **Merging** involves combining two sets of data items according to some criterion.
  - **Traversing** means accessing each element of a data structure at least once. Used particularly on graphs and also referred to as visiting of vertices on a graph.

How to write an Algorithm?

- An algorithms is a backbone for any implementation by some desired language. One can implement the algorithm very effectively if some method is followed in writing the algorithm.
- **Algorithm GREATEST**
  - This algorithm is to find the greatest element in the array A. The array contains n elements. Max will be a variable which stores greatest element of the array and i is index of an array

1. [check whether the array A is empty]
   if n < 1
   Then write ('Array is empty')
   Exit
2. [Initialize the Max with A[1] i.e. first, element of an array]
   Max← A[1] (Considering first element is largest)
   i ← 2
3. [Examine all the elements of an array]
   Repeat through step 5 while i < = n
4. [Check if any other element of array smaller than the next element]
   If Max < A[i]
   Then Max ← A[i]
5. [Check the next element by setting next index]
   i ← i + 1
6. [Finished]
   Exit

How to write an Algorithm?

- **Discussion:**
  - If you have observed the above algorithm, algorithm starts at step 1 and continues a head.
  - At step 2 the variable Max is initialized to A[1], that means we have assumed that very first element of an array is greatest one.
  - Then check whether next element of an array is greatest or not.
  - Thus observe all the elements of an array if any larger element is obtained it will be stored in Max.
  - Thus the variable Max will be over written by larger element of an array.
  - Finally in Max largest element will get stored.

How to write an Algorithm?

- There is a specific format of writing the algorithm.
  1. Every algorithm is given an identifying name and the name should be in capitals. In the above we have given GREATEST as a name of algorithm
  2. Algorithm name should be followed by a brief description of that algorithm. The description may also include the variables used in algorithm. Even if some assumptions are made for that algorithm, then those can be included here.
  3. The actual algorithm is made up of sequence of steps which should be numbered. The description of that step should be written in square brackets. Then actual action statements should be written.
  4. If comments are to be given in that algorithm it should be given in round parenthesis, the comments are given for better understanding of an algorithm.

How to write an Algorithm?

5. Assignment statements are given with the help of ←, in our sample program Max ← A[i]. We can use := symbol for assignment statement.
6. If statement is given as

   If condition

   Then ----

   The if condition can be given in the other way as

   If condition

   Then - - - - -

   .

   else - - - -

   - - - - - -

   .

   - - - - - - -

How to write an Algorithm?

7. **Case statement -** The case statement is used to make a choice among the several alternatives. In general the case statement is written as,

select case (expression)

case value 1:

case value 2:

.

.

.

case value n:

Default:

How to write an Algorithm?

8. **Repeat statement –** The repeat statement is written for looping such as:

   1. Repeat for i=1,2,3,……10
   2. Repeat while i < 10

      read (array i)

      else

      i← i + 1

   Repeat for i = 1,2,3,…….10 while count > 1

9. **Go to and Exitloop statements** – The Go To is unconditional transfer of control to the particular step.

   Go To step 5

   5: - - - - -

How to write an Algorithm?

The Exitloop causes immediate, abrupt exit from loop

For the nested loops Exitloops can be used to come out of it immediately

10. Exit statement is used to terminate an algorithm

For example

If condition then

Exit

11. Arrays can be represented as Array [i1,i2,i3,…..in] where i1,i2,i3,..in are the indices or locations of array.
12. The input and output statements can be given by read (variable name) and write (variable name)

How to write an Algorithm?

13. The sub-algorithms can be given by functions or procedures. Hence while writing function for the algorithm simply write –

    Function name_of_the_function which should be followed by steps of that function,

For example

Function SQUARE (NUM)

In this function we calculate square of variable NUM. And result will be stored in the variable RESULT.

1. [Compute the square of NUM]

   RESULT ← NUM * NM

2. [Return the value of RESULT]

   Return (RESULT)

- In the same way procedure is written. The only thing is we use keyword *procedure name_of_procedure*

Implementation of Algorithm

- Any program can be created with the help of two things **algorithm** and **data structures**.
- To develop any program we should first select a proper data structure and then develop an algorithm for implementing the given problem with the help of a data structure which we ca have chosen.

- This is similar to construction of a house
  - You don't just start house construction
  - Rather you consult an architect and then put ideas and suggestion, accordingly he draw up a plan of the house, then he discusses it with you.
  - If you have suggestions, the architects notes it down and makes the necessary changes in the plan until you are happy.
  - Finally the blue print of the house gets ready and actual construction activity starts. This becomes easy and systematic for construction of the desired house.

Implementation of Algorithm

- We present a technique for the development of program called the program development cycle which involves several steps as follows:
  - Feasibility study
  - Requirement analysis and problem specification.
  - Design
  - Coding
  - Debugging
  - Testing
  - Maintenance.
- Lets discuss each step one by one
  - **Feasibility study:**
    - The problem is analyzed to decide whether it is feasible to develop some program for the given problem statement. If we found that it is feasible then the further steps will be carried out

Implementation of Algorithm

- **Requirement analysis and problem specification**
  - The programmer has to find out the essential requirement for solving the given problem by communicating with the users of the software. Programmer then has to decide what are the inputs needed for the program, in which form the inputs are to be given, the order of the inputs, what kind of output should be generated. Thus the total requirement for the program has to be analyzed.
- **Design:**
  - Once the requirement analysis is done, the design can be prepared using the problem specification document. In this step, the algorithm has to be designed for some most suitable data structure.

Implementation of Algorithm

- **Coding:**
  - When the design of the program is ready then coding becomes a simple job. Because the programming language is selected in the design, coding begins by simply breaking the problem into small modules.
- **Debugging:**
  - in this phase we compile the code and check whether any errors are there. If any error is there then we try to eliminate the errors. The debugging needs complete scan of the program.
- **Testing:**
  - Here, certain set of input data is given to the program and the program should show the desired results as output.

Implementation of Algorithm

- **Maintenance:**
  - Once the code is ready, it is tested properly and in case the users requires some modifications in the code then those modifications should easily be carried out. Modularity in the code has to be maintained.
- **Documentation:**
  - The documentation is not a separate step in program development process but it is required at every step. Documentation means providing help or some manual which will help the user to make use of the code in proper direction.
  - It is a good practice to maintain some kind of document for every phase of compilation process.

Algorithmic Strategies

- Algorithmic strategies is a general approach by which many problems can be solved algorithmically
- Algorithmic strategies are also called as **algorithmic techniques.**
- Various algorithmic strategies are-
  - **Divide and conquer** – In this strategy the problem is divided into smaller sub-problems and these sub-problems are solved to obtain the solution to main problem.
  - **Dynamic Programming-** The problem is divided into smaller instances and results of smaller reoccurring instances are obtained to solve the problem.
  - **Greedy technique-** From a set of obtained solutions the best possible solution is chosen each time, to obtain the final solution.
  - **Back tracking-** In this method, in order to obtain the solution trial and error method is followed.