Introduction to Data Structures

Module 1

Data

- Data can be numeric or character
- When we process this data, it becomes information.

- Eg- MH32V2126,
  - A string, sequence of character type data
  - But when it represents the car registration number then it is processed data
  - I.e. information.

DATA STRUCTURES

Data may be organized in many different ways. The logical or mathematical model of a particular organization of data is called a Data Structure.

Or

In other words, a Data Structure is a way of organizing all the data items that considers not only the elements stored but also the relationship between them.

Data Structure

- Data Structure is a way to organize the data in some way so we can perform operations on these data in effective ways.
- Eg-List, Stack, Queue, Tree

Selection/choice of data structures

- The choice of Data Structure model depends on two considerations-

1) It must be rich enough in structure to mirror the actual relationships of the data in the real world.

2) The structure should be simple enough that one can effectively process the data when necessary,

i.e. the type of operation to be performed.

Data Structure

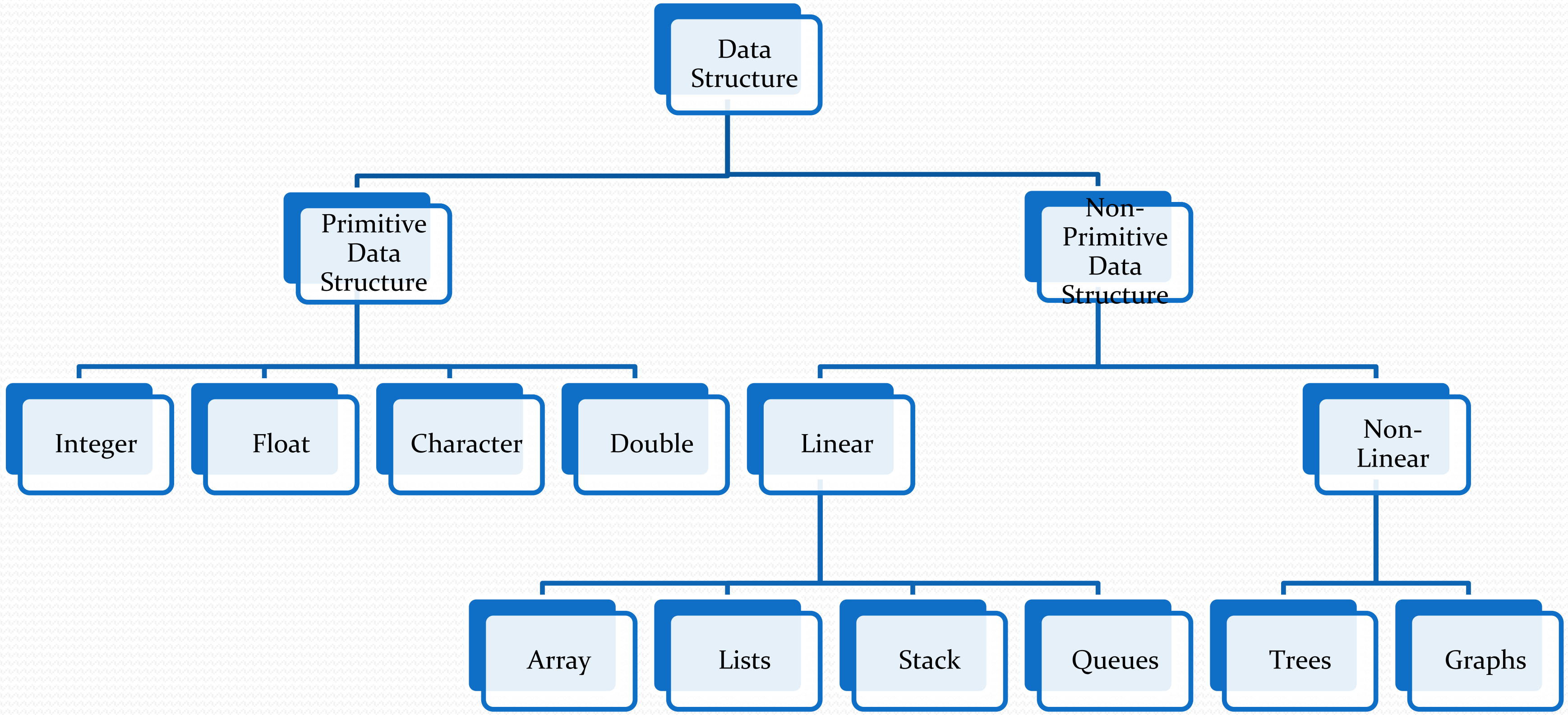
- Data Structures affects the design of both the structural and functional aspects of a program.

Data Structure

ALGORITHM+DATA STRUCTURE=PROGRAM

- Algorithm is a step by step procedure to solve a particular problem.
- To develop a program we should select an appropriate data structure for that algorithm.
- Thus , algorithm and its associated data structure form a program.

CLASSIFICATION OF DATA STRUCTURES



PRIMITIVE DATA STRUCTURES

- These are basic data structures and are directly operated upon by the machine level instructions.
- These in general have different representations on different computers.
- Integers, floating point numbers, character constants, double etc fall in this category.

NON-PRIMITIVE DATA STRUCTURES

- These are more sophisticated data structures.
- These are derived from the basic data types.
- The non-primitive data structures emphasize on structuring of a
  - group of homogenous(same type) or
  - heterogenous (different type) data items.
- Eg- Arrays, lists, trees etc.

NON-PRIMITIVE DATA STRUCTURES

- Classified as
  - Linear data structures
  - Non- Linear data structures

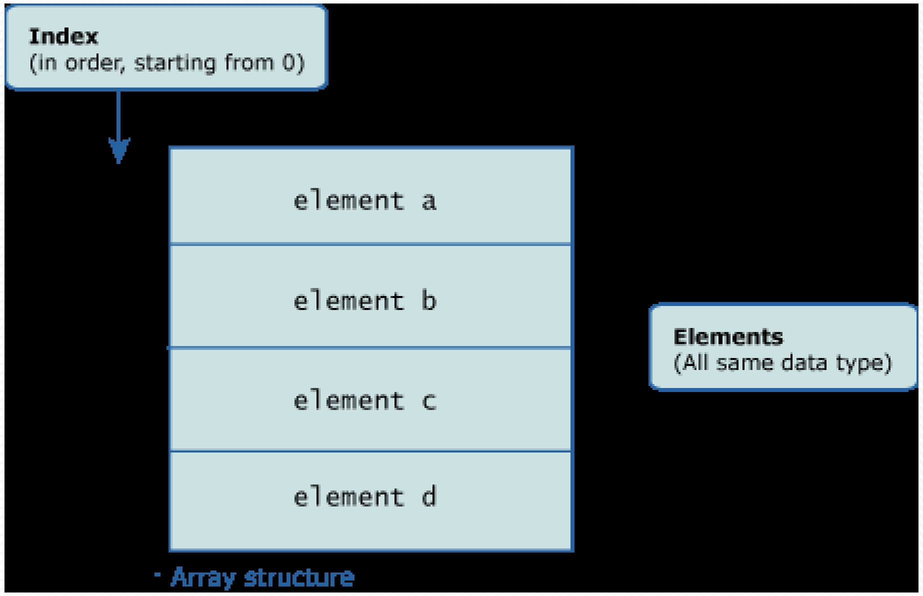
Linear Data Structures

- A data structure is said to be linear
  - if its elements form a sequence or a linear list.
- Here elements are arranged in one dimension or linear fashion.
- All one-one relation can be handled through linear data structures.
- Eg- Lists, stacks and queues

Arrays

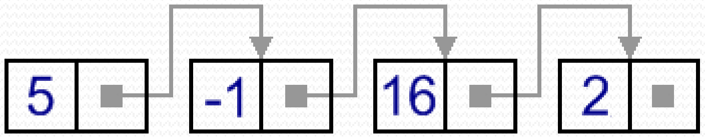
- An array can be defined as a collection of homogenous data type.
- An array can store either all integers, all floating point numbers, all characters or any other complex data type.
- Declaration of an integer array is as follows-

int a[10];



Lists

- Linked lists are special lists of some data elements linked to one another.
- The logical ordering is represented by
  - having each element pointing to the next element.
- Each element is called a node, which has two parts, INFO part which stores the information and pointer which points to the next element.



Stacks

Stacks

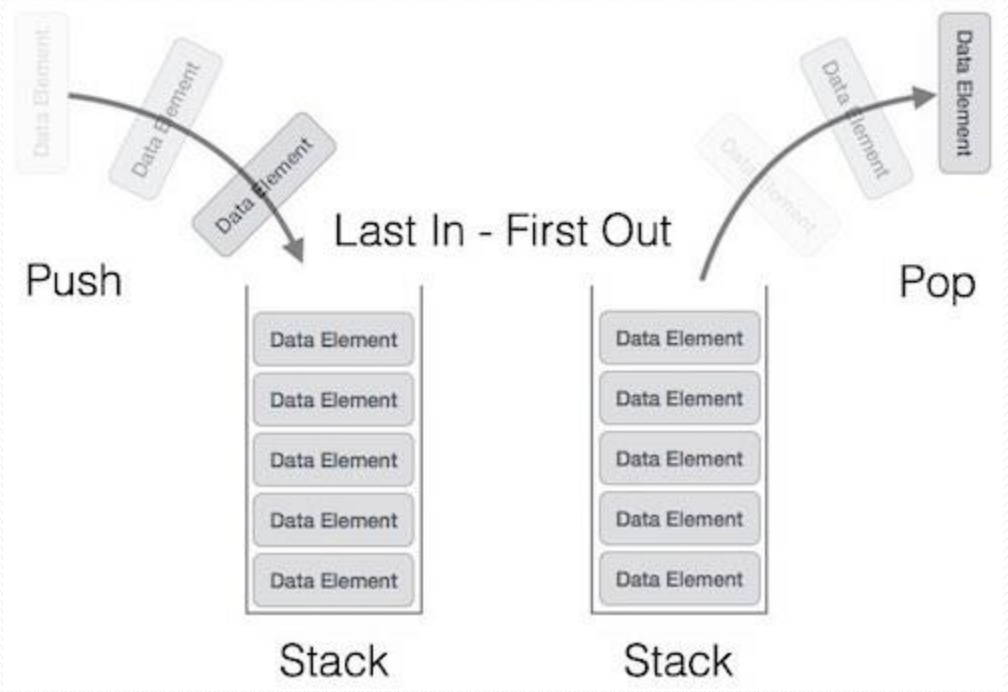
- It could be thought of just like a stack of plates placed on table in a party,
  - a guest always takes off a fresh plate from the top, and
  - the new plates are placed on the top of the stack .

Stacks

- The plate which is at the top is the first one to be removed,
- The plate which has been placed at the bottommost position remains in the stack for the longest period of time.
- So, it can be simply seen to follow **LIFO(Last In First Out)/FILO(First In Last Out) order.**

Stacks

- Stack is a linear data structure which follows a particular order in which the operations are performed.
- The order may be LIFO(Last In First Out) or FILO(First In Last Out).



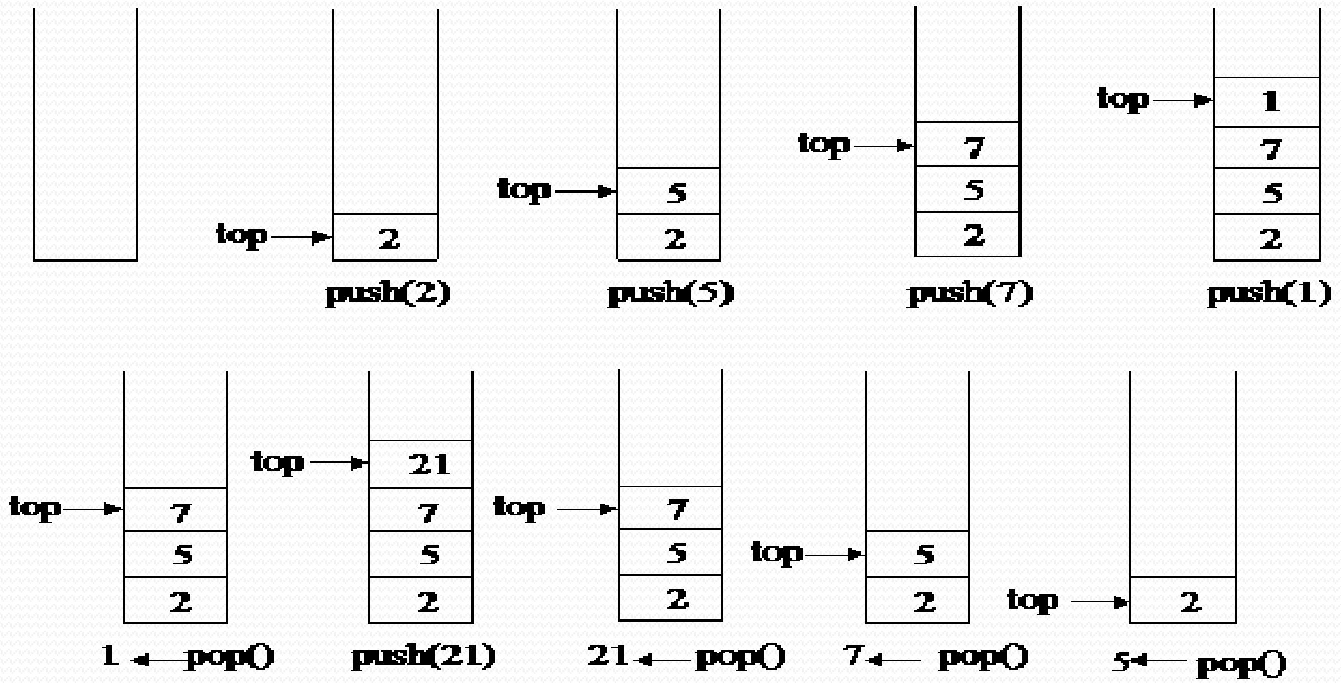
Stacks

- A Stack is an ordered collection of elements , 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).

Operation on Stacks

- **Push operation**-The insertion of an element into stack
- **Pop operation** - Deletion of an element from the stack

- In stack we always keep track of the last element present in the list with a pointer called top.

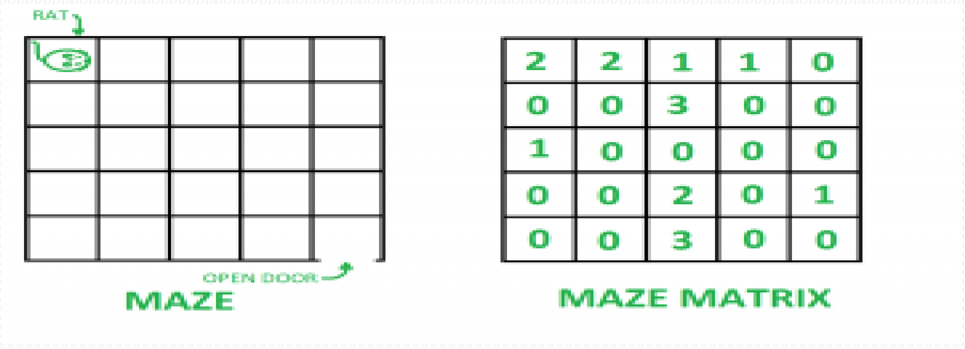


Applications of stack:

- Expression evaluation and syntax parsing
- Balancing of symbols, paranthesis
- Infix to Postfix /Prefix conversion
- Reverse a String using Stack
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.

Applications of stack:

- **Backtracking**
  - Backtracking is an algorithmic-technique for
    - solving problems recursively by trying to build a solution incrementally, one piece at a time,
    - removing those solutions that fail to satisfy the constraints of the problem at any point of time
  - Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver



2	2	1	1	0
0	0	3	0	0
1	0	0	0	0
0	0	2	0	1
0	0	3	0	0

- **Backtracking-**Finding the correct path in a maze.
  - There are a series of points, from the starting point to the destination.
  - We start from one point. To reach the final destination, there are several paths.
  - Suppose we choose a random path. After following a certain path, we realise that the path we have chosen is wrong. So we need to find a way by which we can return to the beginning of that path.

- **Backtracking-**Finding the correct path in a maze.
  - This can be done with the use of stacks. With the help of stacks, we remember the point where we have reached.
  - This is done by pushing that point into the stack.
  - In case we end up on the wrong path, we can pop the top point from the stack and thus return to the last point and continue our quest to find the right path.

Applications of stack:

- **Depth-first search**
  - The prototypical example of a backtracking algorithm is depth-first search, which finds all vertices of a graph that can be reached from a specified starting vertex.

- In Graph Algorithms like Topological Sorting and Strongly Connected Components

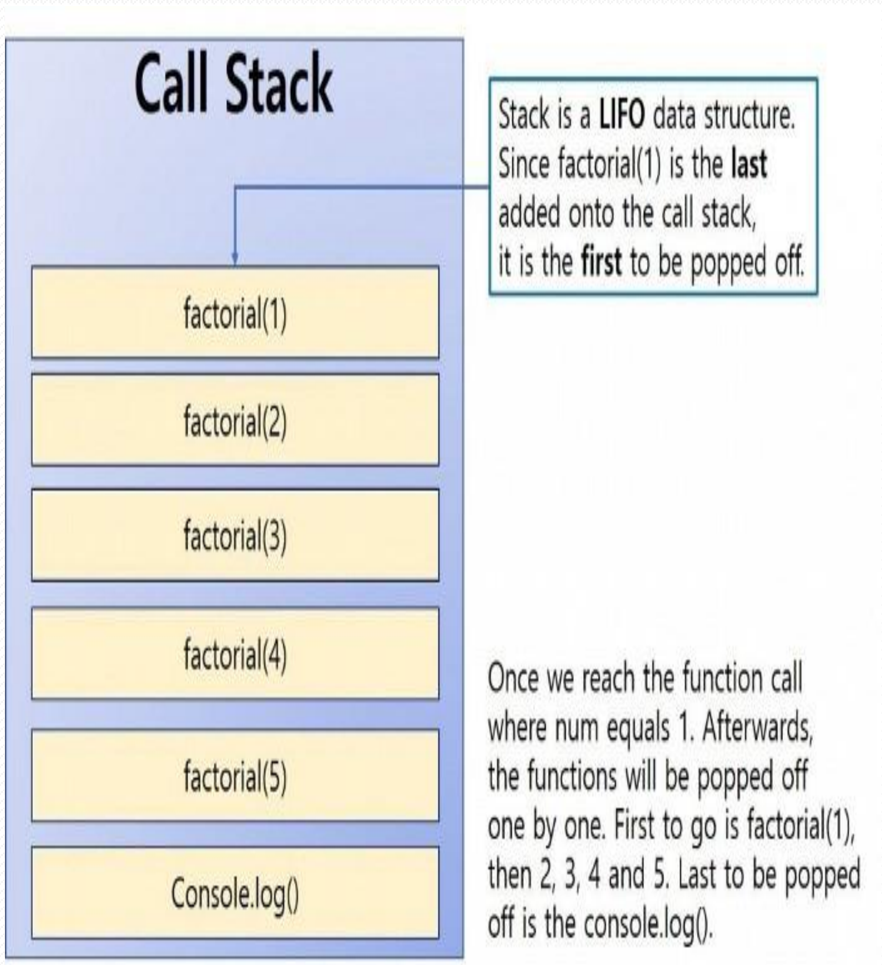
Applications of stack:

- **Compile time memory management**
- Almost all calling conventions

Function calls using stack

- **Compile time memory management**
  - A number of programming languages are stack-oriented,
    - meaning they define most basic operations as taking their **arguments from the stack, and placing any return values back on the stack.**
    - Eg- adding two numbers, printing a character

Function calls using stack

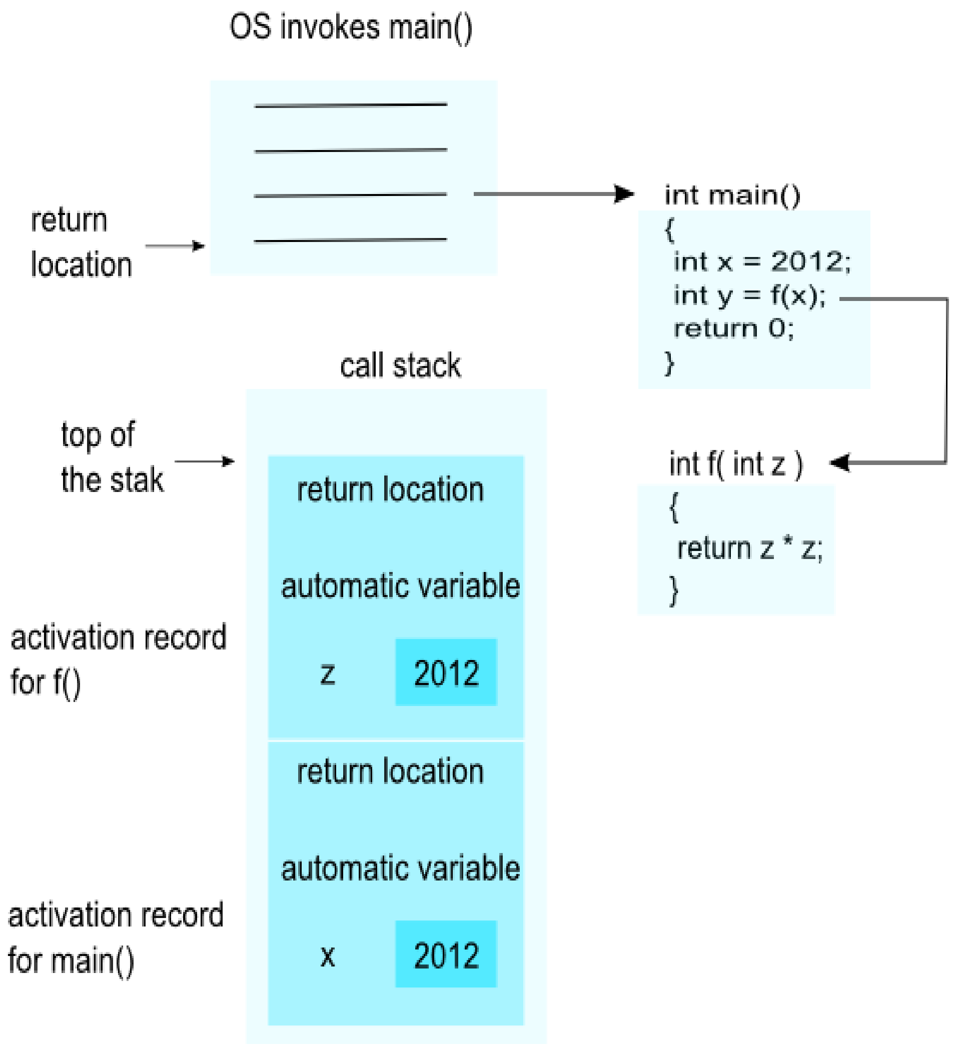


Stack is a **LIFO** data structure. Since factorial(1) is the **last** added onto the call stack, it is the **first** to be popped off.

Once we reach the function call where num equals 1. Afterwards, the functions will be popped off one by one. First to go is factorial(1), then 2, 3, 4 and 5. Last to be popped off is the console.log().

- **Stacks are an important way of supporting nested or recursive function calls.**

Function calls using stack

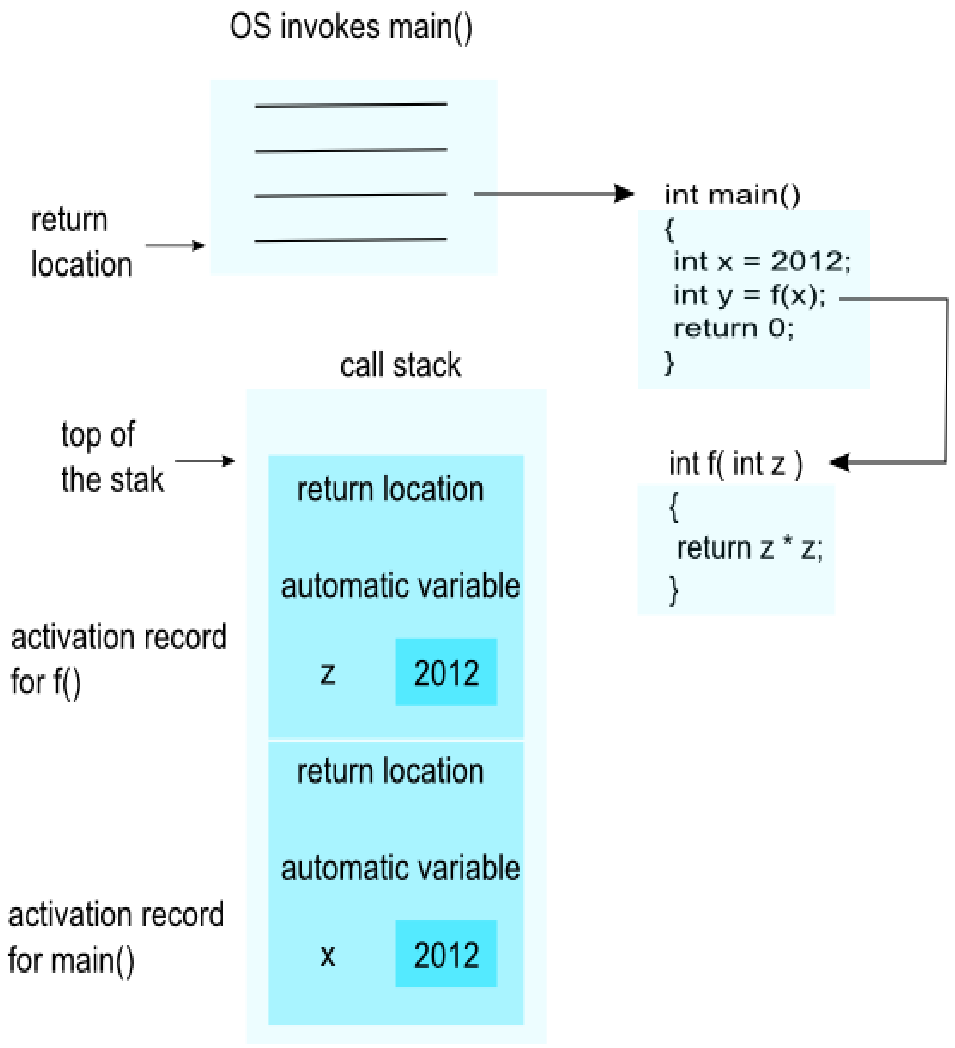


```
int main()
{
 int x = 2012;
 int y = f(x);
 return 0;
}
```

```
int f( int z )
{
 return z * z;
}
```

- The ways in which subroutines receive their parameters and return results—use a special stack (the "call stack")

Function calls using stack



```
int main()
{
 int x = 2012;
 int y = f(x);
 return 0;
}
```

```
int f( int z )
{
 return z * z;
}
```

- Call stack holds information
- about procedure/function calling and nesting
- in order to switch to the context of the called function and
- restore to the caller function when the calling finishes.

Applications of stack:

- **Efficient algorithms**
  - The nearest-neighbor chain algorithm,
  - Graham scan, an algorithm for the convex hull of a two-dimensional system of points. A convex hull of a subset of the input is maintained in a stack, which is used to find and remove concavities in the boundary when a new point is added to the hull

- **Security**
  - Some computing environments use stacks in ways that may make them vulnerable to security breaches and attacks.
  - Programmers working in such environments must take special care to avoid the pitfalls of these implementations.

Data Structures | Stack | Question 2

Which one of the following is an application of Stack Data Structure?

(A) Managing function calls
(B) The stock span problem
(C) Arithmetic expression evaluation
(D) All of the above

Data Structures | Stack | Question 2

Which one of the following is an application of Stack Data Structure?
(A) Managing function calls
(B) The stock span problem
(C) Arithmetic expression evaluation
(D) All of the above

Answer: (D)

Queues

Real life Examples of Queue

- The people standing in a railway reservation row are an example of queue. Each new person comes and stands at the end of the row (rear end of queue) and persons getting their reservations confirmed , get out of the row from the front end.

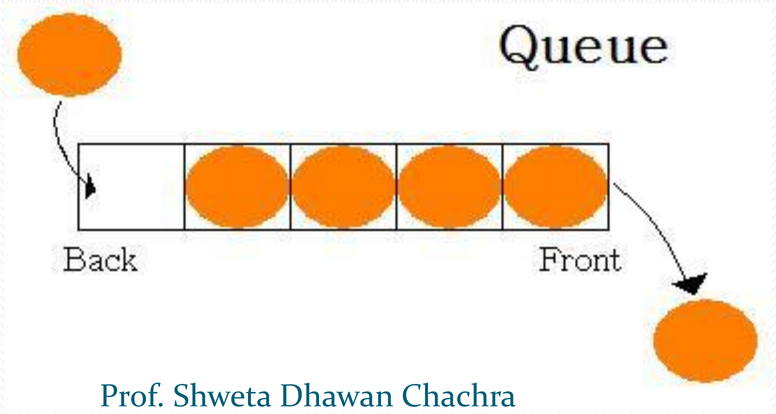
Real life Examples of Queue

- Our software queues have counterparts in real world queues.
- We wait in queues
  - to buy pizza,
  - to enter movie theaters,
  - to drive on a turnpike, and
  - to ride on a roller coaster.

LAST IN
LAST OUT
ONE WAY
FIRST IN
FIRST OUT

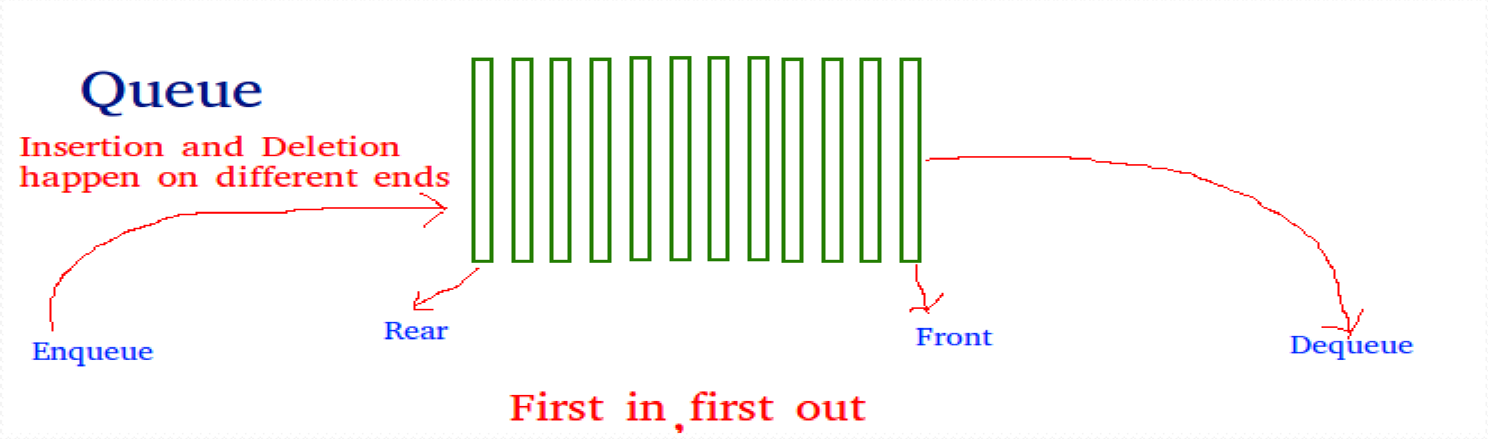
Queues

- A Queue is a linear structure which follows a particular order in which the operations are performed.
- The order is FIFO, Queues are first in first out type of data structures(FIFO).
- In a queue new elements are added to the queue from one end called Rear end and the elements are always removed from the other end called the Front end.



Operations on Queue:

- **Enqueue:**
  - Adds an item to the queue
  - Enter or Insert an item from Rear end
- **Dequeue:**
  - Removes an item from the queue.
  - Service or Delete From the **Front end**



Applications of Queue:

- Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
- **Disk Scheduling**

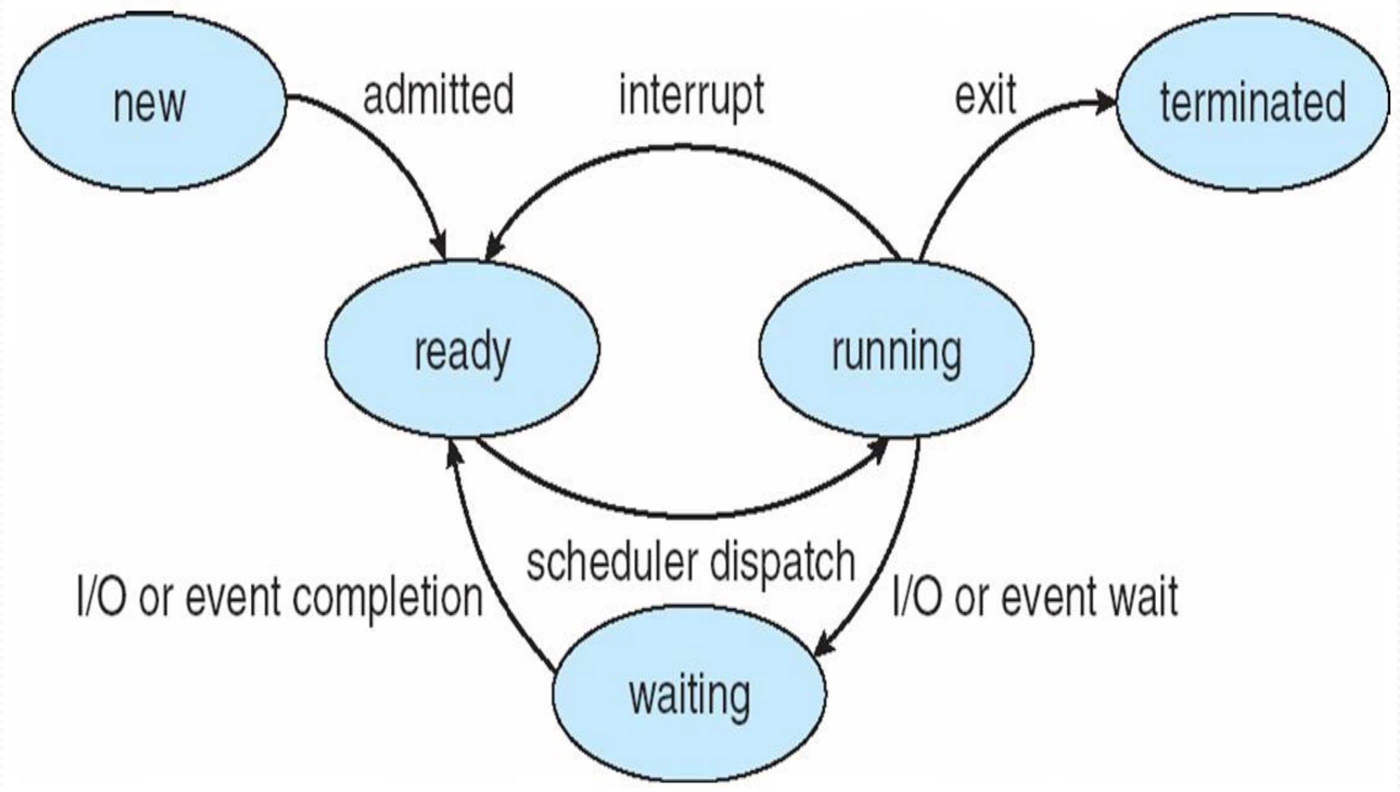
Applications of Queue:

- **CPU scheduling-**

When multiple processes require CPU at the same time, various CPU scheduling algorithms are used which are implemented using Queue data structure.

Applications of Queue:

- **CPU scheduling-**



Applications of Queue:

- When things don’t have to be processed immediately, but have to be processed in First In First Out order like **Breadth First Search**.
- When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include **IO Buffers, pipes, file IO, etc.**
- Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.

Courtesy:https://www.javatpoint.com/data-structure-queue

Queue Example

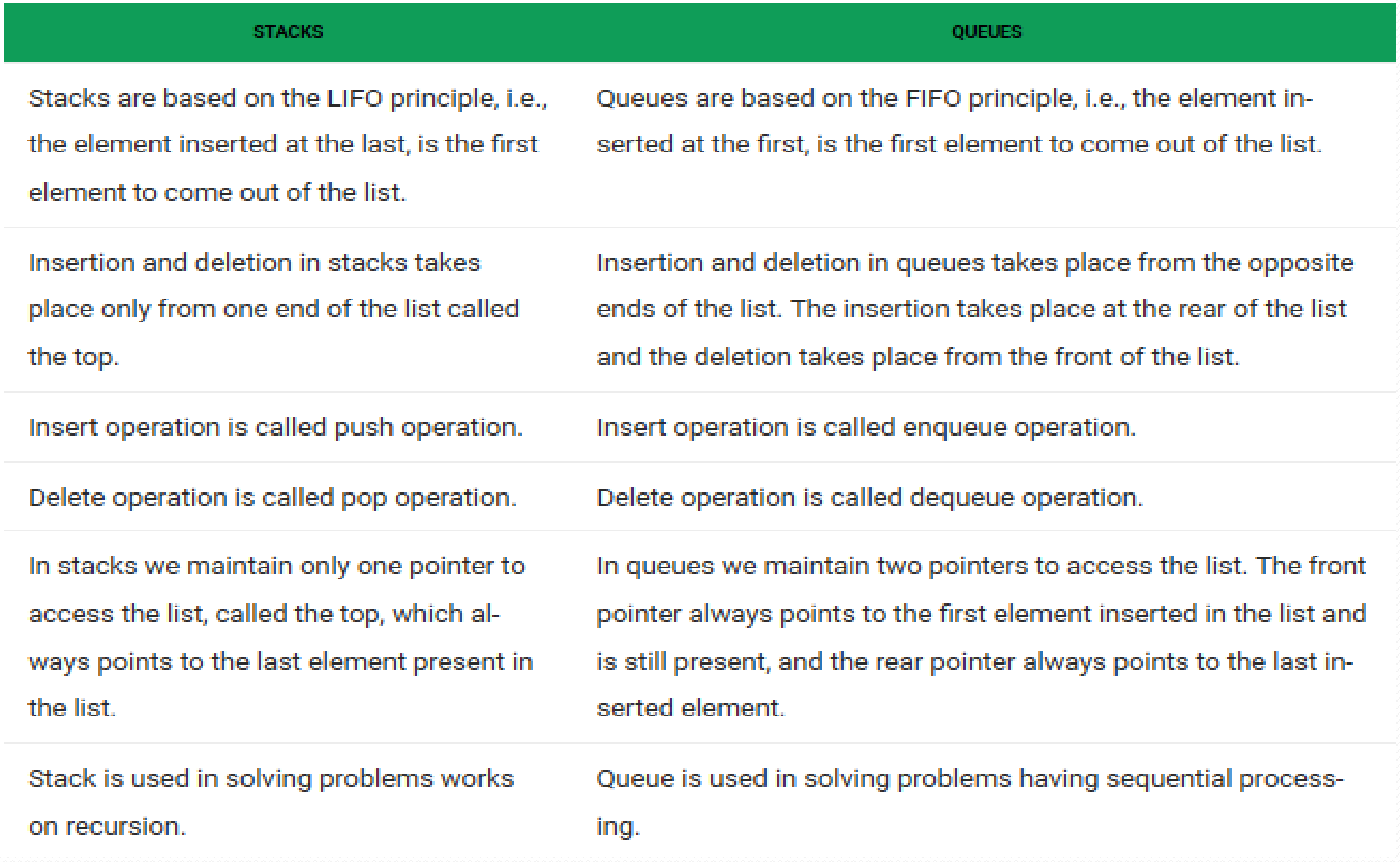
Accessing printer in multiuser environment-

- If a printer is in process and more than one user wants to access the printer then
- it maintains the queue for user requesting access and serves in FIFO manner for giving access.

Difference between Stacks and Queues

- The difference between stacks and queues is in removing.
  - In a stack, we remove the item which was most recently added;
  - In a queue, we remove the item which was least recently added.

Difference between Stacks and Queues



STACKS	QUEUES
Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list.	Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.
Insertion and deletion in stacks takes place only from one end of the list called the top.	Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list.
Insert operation is called push operation.	Insert operation is called enqueue operation.
Delete operation is called pop operation.	Delete operation is called dequeue operation.
In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list.	In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element.
Stack is used in solving problems works on recursion.	Queue is used in solving problems having sequential processing.

Non-Linear Data Structures

- A data structure is said to be non-linear if traversal of nodes is nonlinear in nature.

- Data elements are not arranged sequentially or linearly are called **non-linear data structures**.

- In a non-linear data structure, single level is not involved. Therefore, we can't traverse all the elements in single run only.

Non-Linear Data Structures

- A non-linear data structure has no set sequence of connecting all its elements and each element can have multiple paths to connect to other elements.

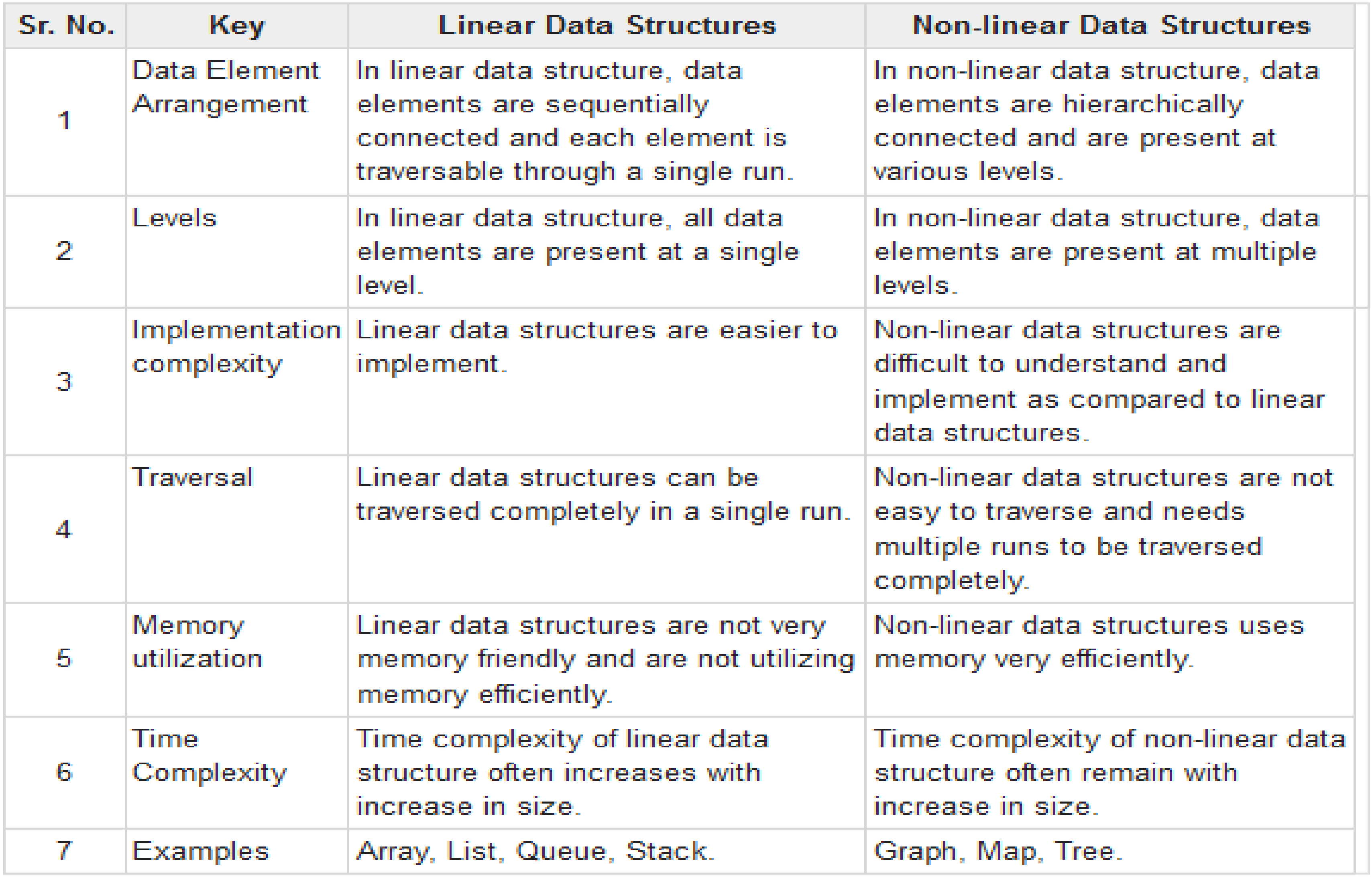
Non-Linear Data Structures

- All one-many, many-one or many-many relations are handled through non-linear data structures.

- Every data element can have a number of predecessors as well as successors.

- Trees, graphs and tables are its examples.

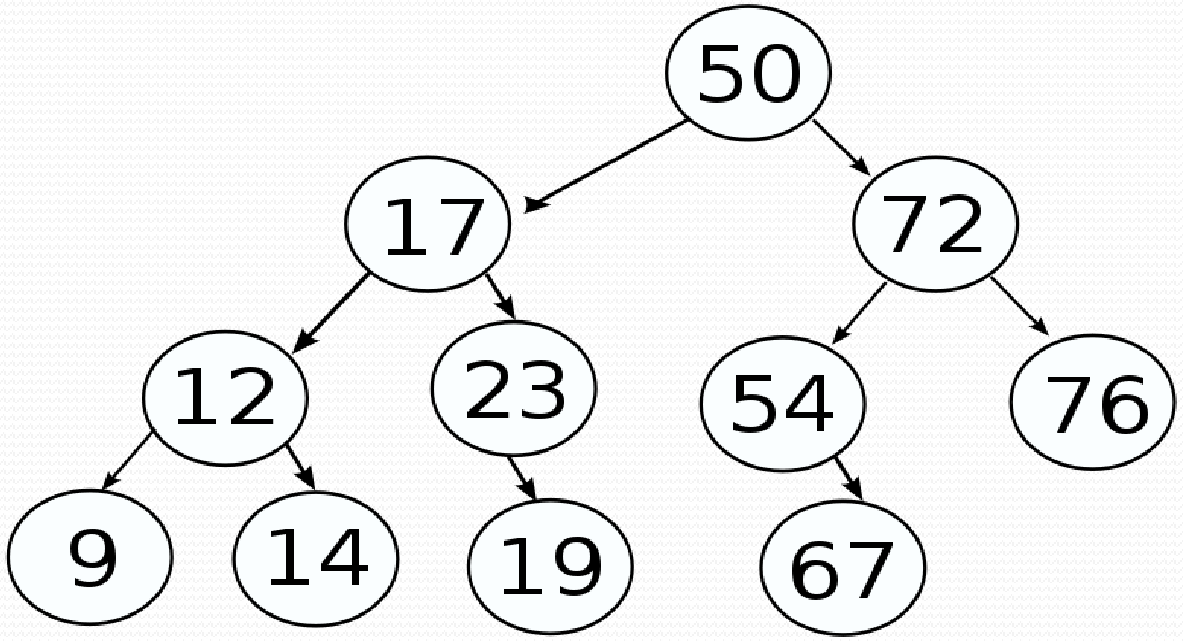
Linear Vs Non Linear Data Structures



Sr. No.	Key	Linear Data Structures	Non-linear Data Structures
1	Data Element Arrangement	In linear data structure, data elements are sequentially connected and each element is traversable through a single run.	In non-linear data structure, data elements are hierarchically connected and are present at various levels.
2	Levels	In linear data structure, all data elements are present at a single level.	In non-linear data structure, data elements are present at multiple levels.
3	Implementation complexity	Linear data structures are easier to implement.	Non-linear data structures are difficult to understand and implement as compared to linear data structures.
4	Traversal	Linear data structures can be traversed completely in a single run.	Non-linear data structures are not easy to traverse and needs multiple runs to be traversed completely.
5	Memory utilization	Linear data structures are not very memory friendly and are not utilizing memory efficiently.	Non-linear data structures uses memory very efficiently.
6	Time Complexity	Time complexity of linear data structure often increases with increase in size.	Time complexity of non-linear data structure often remain with increase in size.
7	Examples	Array, List, Queue, Stack.	Graph, Map, Tree.

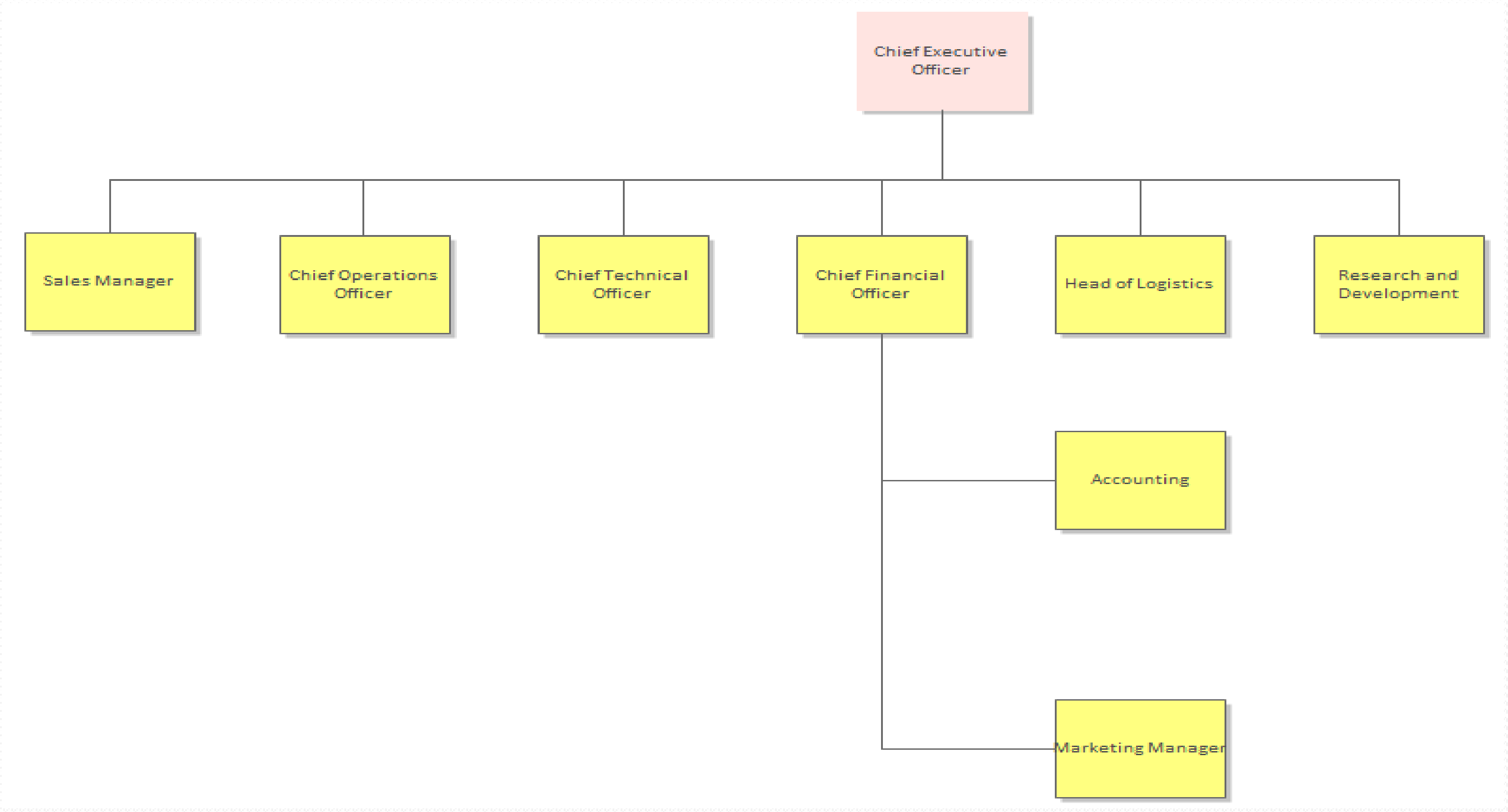
Trees

- A tree is a non- linear data structure used to represent hierarchical relationship existing among several data items.



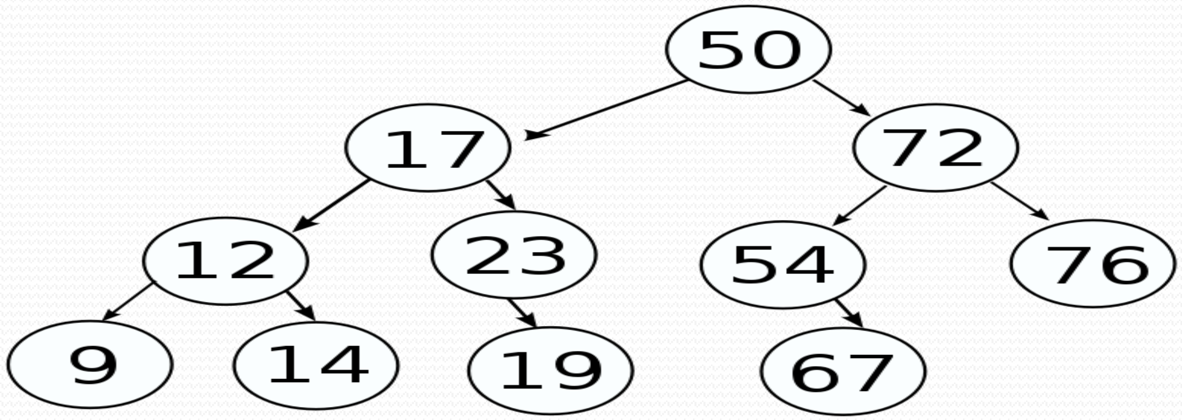
Tree

- Represents Hierarchy
- For eg- The organization structure of an Corporation



Trees

- It is a finite set of one or more data items such that-
  - There is a special data item called the root of the tree.
  - Its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.



Applications of Tree Data Structure

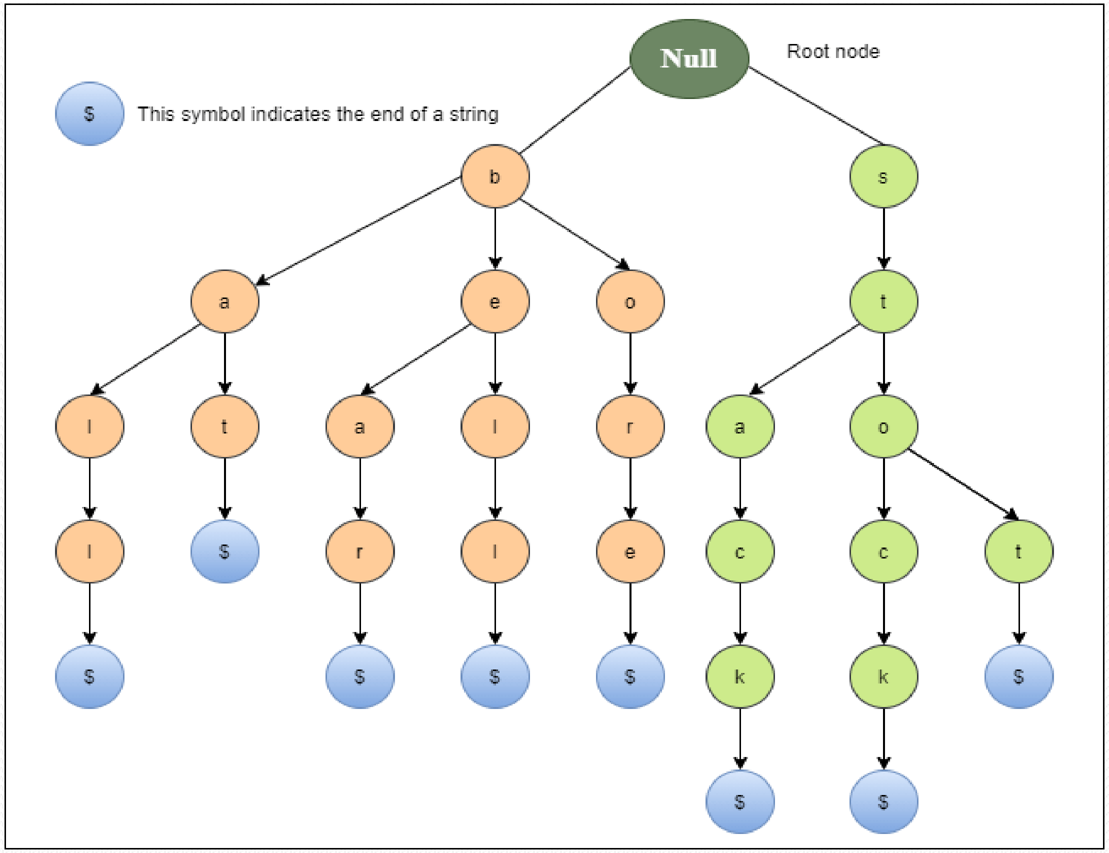
- Store hierarchical data, like folder structure, organization structure, XML/HTML data.
- Binary Search Tree is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item
- Heap is a tree data structure which is implemented using arrays and used to implement priority queues.

Applications of Tree Data Structure

- B-Tree and B+ Tree : They are used to implement indexing in databases.
- Syntax Tree: Used in Compilers.
- Spanning Trees and shortest path trees are used in routers and bridges respectively in computer networks

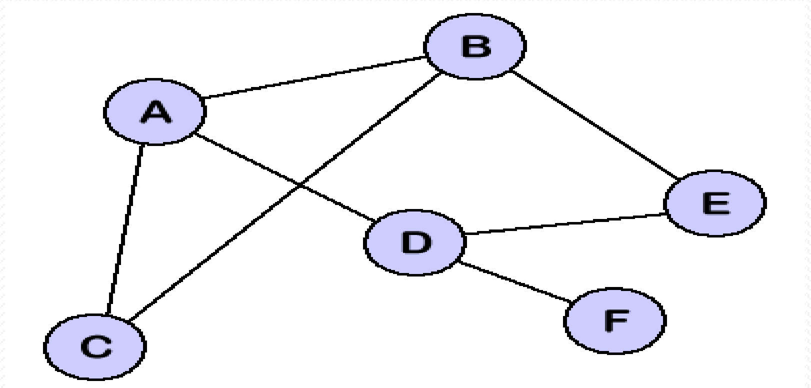
Applications of Tree Data Structure

- Trie : Used to implement dictionaries with prefix lookup.



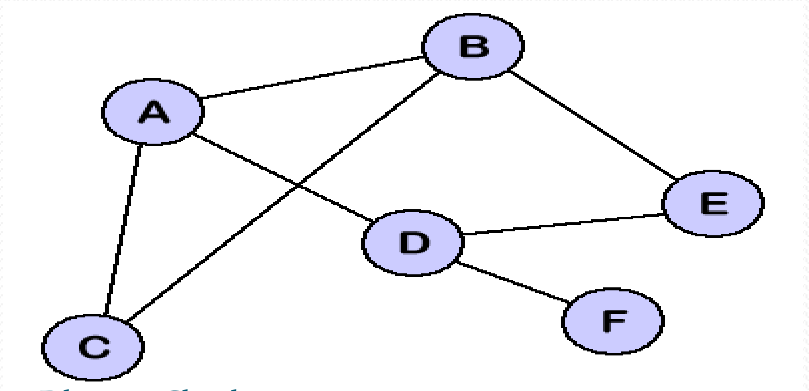
Graphs

- Graph is a non-linear data structure capable of representing many kinds of physical data structures.
- It has found its application in diverse fields like Geography, Chemistry and Engineering Sciences.



Graphs

- A graph G(V,E) is a set of vertices V and a set of edges E.
  - An edge connects a pair of vertices and many have weight such as length, cost etc.
  - Vertices =point or circles
  - Edges=arcs or line segment.
  - Edge E=(V,W) where V and W are pair of vertices.



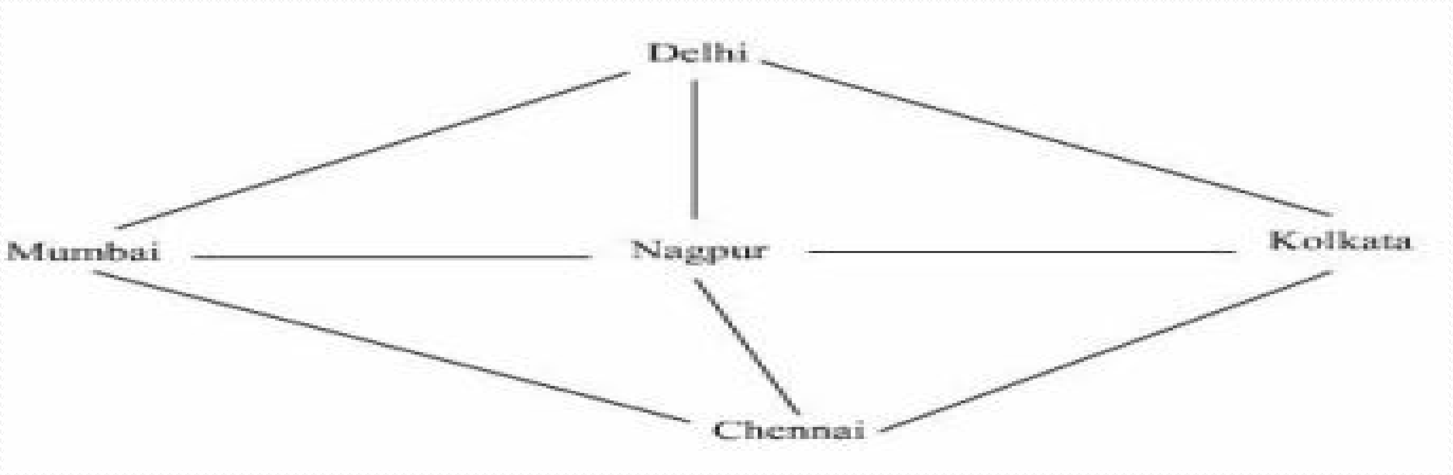
GRAPHS Representation :

- Graph representation of a **road network**
- A road network is a simple example of a graph,
- Vertices represents **cities and road** connecting them are correspond to edges.

GRAPHS Representation :

V= { Delhi, Chenai, Kolkata, Mumbai, Nagpur }

E= { (Delhi, Kolkata), (Delhi, Mumbai}, (Delhi, Nagpur), (Chenai, Kolkata), (Chenai, Mumbai), (Chenai, Nagpur), (Kolkata, Nagpur), (Mumbai, Nagpur) }



Applications of Graph Data Structure

- ?

Applications of Graph Data Structure

- Google maps
- Facebook
- World Wide Web
- Operating System

Applications of Graph Data Structure

- Google maps –
  - ?

Applications of Graph Data Structure

- In Computer science graphs are used to represent the flow of computation.
- Google maps –
  - uses graphs for building transportation systems,
  - where intersection of two(or more) roads are considered to be a vertex and
  - the road connecting two vertices is considered to be an edge,
  - thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.

Applications of Graph Data Structure

- In Facebook,
  - ?

Applications of Graph Data Structure

- In Facebook,
  - users are considered to be the vertices and
  - if they are friends then there is an edge running between them.
  - Facebook’s Friend suggestion algorithm uses graph theory.
  - Facebook is an example of undirected graph.

Applications of Graph Data Structure

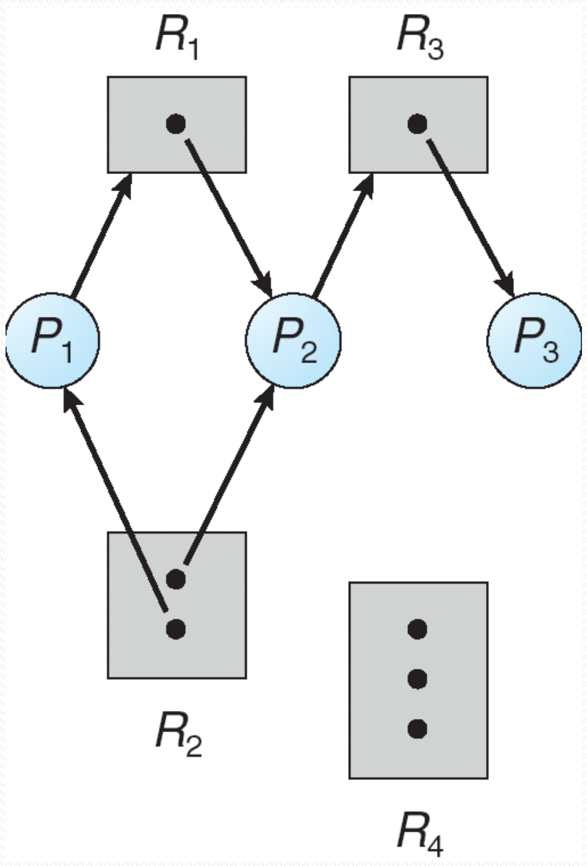
- In World Wide Web,
  - web pages are considered to be the vertices.
  - There is an edge from a page u to other page v if there is a link of page v on page u.
  - This is an example of Directed graph.
  - It was the basic idea behind Google Page Ranking Algorithm.

Applications of Graph Data Structure

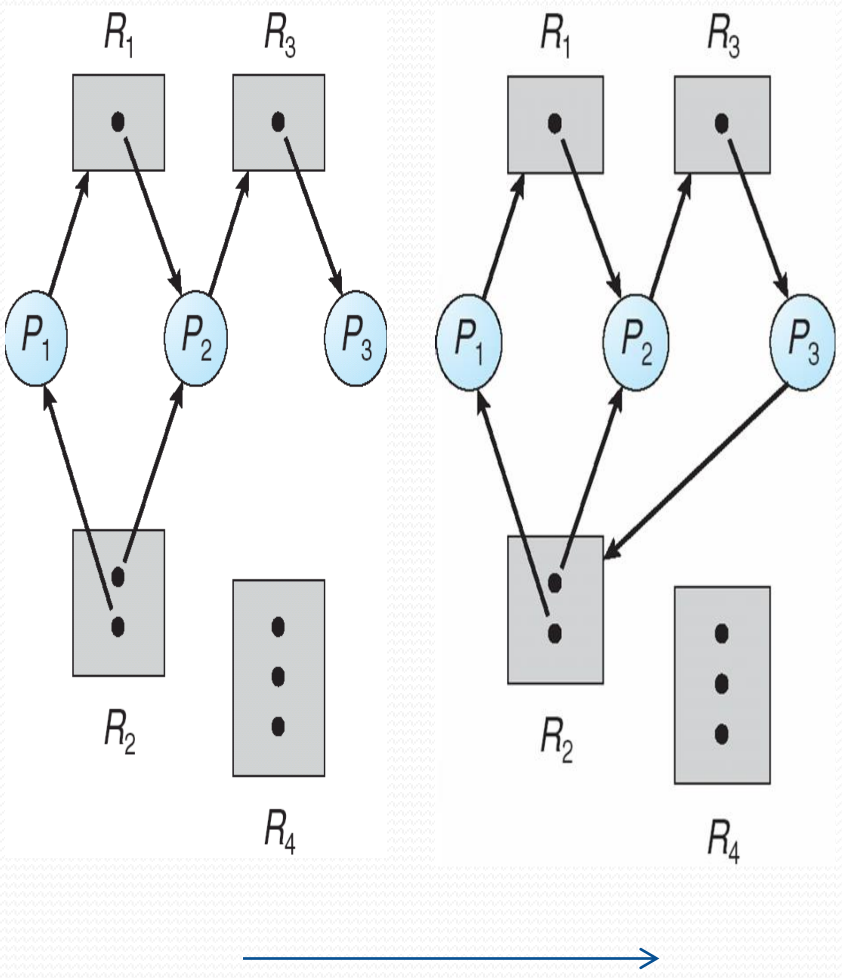
- In Operating System,
  - we come across the **Resource Allocation Graph** where
    - **each process and resources are considered to be vertices.**

Applications of Graph Data Structure

- Edges are drawn from
  - resources to the **allocated** process, or
  - from **requesting** process to the requested resource.
- If this leads to any **formation of a cycle then a deadlock may occur.**



Resource Allocation Graph With A Deadlock



- Suppose P3 requests for R2,
- Since no resource instance is free, a request edge P3->R2 is added
- So, Now Two cycles exist –

P1->R1->P2->R3->P3->R2->P1

P2->R3->P3->R2->P2

- P1,P2,P3 are deadlocked

GATE CS 2004

The best data structure to check whether an arithmetic expression has balanced parentheses is a (GATE CS 2004)

a) queue
b) stack
c) tree
d) list

GATE CS 2004

The best data structure to check whether an arithmetic expression has balanced parentheses is a (GATE CS 2004)

a) queue
b) stack
c) tree
d) list

Answer(b)