MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-1

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Definition, ADT, Types of Data Structures- Linear & Non Linear Data Structures

Introduction :

- Data structure defines a way of organizing all data items that consider not only the elements stored but also stores the relationship between the elements.
- Data structure can divide with two types: linear and non-linear structure.
- Data structures are framework for organizing and storing information in virtual memory forms.
- Determine the various types of abstract data such as queue, stack, lists and deque,ADT.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Data
- Linear data structures
- Non-linear data structures

Detailed content of the Lecture:

- Data structure in C programming language is a specialized format for organizing and storing data.
- In general data structure types include the file, array, record, table, tree.etc..
- Data structure introduction refers to a scheme for organizing data, or in other words it is an arrangement of data in computer's memory in such a way that it could make the data quickly available to the processor for required calculations.

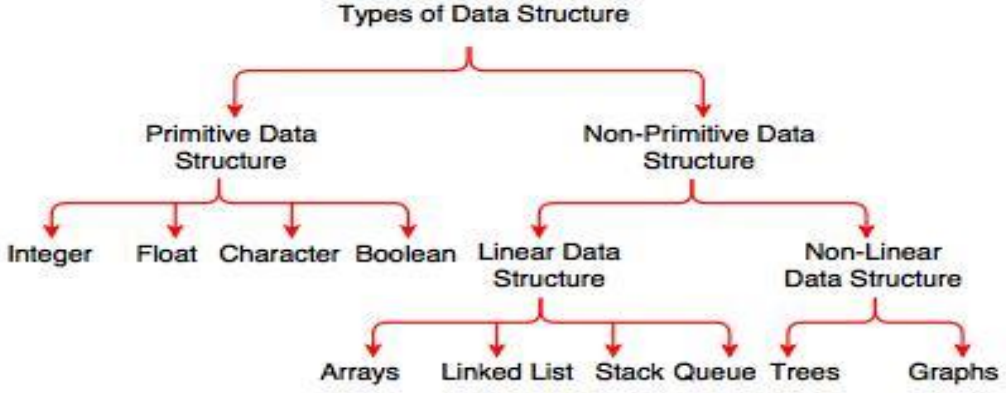
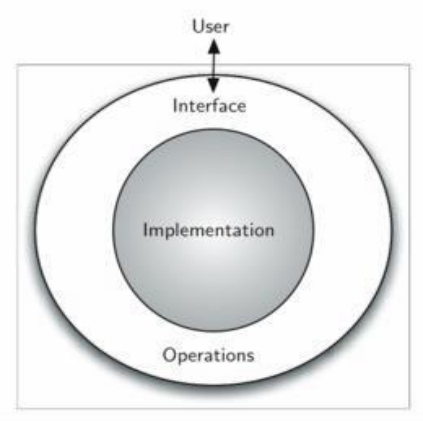


Fig. Types of Data Structure

- As data structure is a scheme for data organization so the functional definition of a data structure should be independent of its implementation.
- Data structures can be classified as Simple data structure,Compound data structure, Linear data structure, Non linear data structure.
- Linear data structures are data structures having a linear relationship between its adjacent elements. Linked lists are examples of linear data structures.eg linked list, array
- Non-linear data structure can be constructed as a collection of randomly distributed set of data item joined together by using a special pointer (tag). In non-linear Data structure the relationship of adjacency is not maintained between the data items.
- The simplest type of data structure is a linear array. In computer science, an array data structure or simply an array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key.

Abstract Data Type

- An abstract data type is a set of operations for which the implementation of the data structure is not specified anywhere in the program.



Video Content / Details of website for further learning (if any):

https://youtu.be/zWg7U0OEAoE

https://nptel.ac.in/courses/106/102/106102064/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012- **Page Nos:** 1-6

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-2**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Array: Representation ofarrays, structure and Pointers, Applications of arrays

Introduction :

An array data structure, or simply an array, is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key.

- They are also used to implement many other data structures, such as lists and strings.
- It occupies less memory than a linked list for the same number of elements.

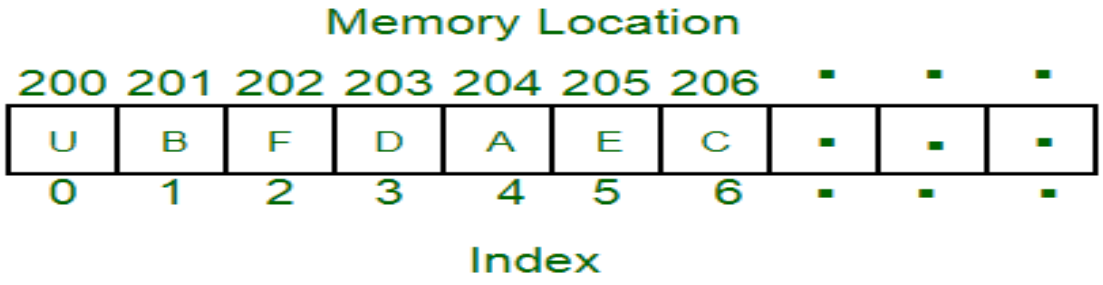
Prerequisite knowledge for Complete understanding and learning of Topic:

- Memory allocation
- Data types
- Linear data structures

Detailed content of the Lecture:

Array:

- Array is a collection of variables belongings to the same data type. You can store group of data of same data type in an array.
- Array might be belonging to any of the data types
- Array size must be a constant value.Always, Contiguous (adjacent) memory locations are used to store array elements in memory.



- One dimensional is also known as single dimensional array where the elements be accessed in sequential order. This type of array will be accessed by the subscript of either a column or row now.

- When the number of dimensions specified is more than one then it is called multi dimensional array. It includes 2D and 3D arrays.
- The size of the array is fixed. Most often this size is specified at compile time. This makes the programmers to allocate arrays, which seems "large enough" thanrequired.
- Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to makeroom.
- Deleting an element from an array is not possible. Linked lists have their own strengths and weaknesses, but the happen to be strong where arrays areweak.
- Generally array's allocates the memory for all its elements in one block whereas linked lists use an entirely different strategy.
- Linked lists allocate memory for each element separately and only whennecessary.

APPLICATIONS OF ARRAYS

- Arrays are used to implement mathematical vectors and matrices, as well as otherkinds of rectangular tables.
- Arrays are used to implement other data structures, such as lists, heaps, hash tables, dequeues, queues and stacks.
- Arrays are also used to implement CPU Scheduling 8lgorithms

Video Content / Details of website for further learning (if any):

https://www.coursera.org/lecture/data-structures/arrays-OsBSF

https://nptel.ac.in/courses/106105085/

Important Books/Journals for further learning including the page nos.:

E.Horowitz,S.Sahni Susan ,Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008- **Page Nos:7,8**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-3

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Structure and Pointer

Introduction :

- Structure is a collection of variables belongings to the different data type. You can store group of data of different data type in an array.
- A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Pointer
- Memory address
- Data types

Detailed content of the Lecture:

Structure

- A structure creates a data type that can be used to group items of possibly different types into a single type. 'struct' keyword is used to create a structure.
- Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.The structure variables can be a normal structure variable or a pointer variable to access the data.
- An array of structures is simply an array in which each element is a structure of the same type it referencing and subscripting of these arrays follow the same rules as simple arrays.

Syntax

```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   data_type memeber;
};
```

Advantages

- It can hold variables of different data types.
- create objects containing different types of attributes.
- It allows us to re-use the data layout across programs.

Application of structure

- It is used to implement other data structures like linked lists, stacks, queues, trees, graphs etc Clearing screen
- Adjusting Cursor Position
- Drawing any graphics shape on the screen
- Receiving a key from the keyboard
- Finding out the list of equipment attached to the computer

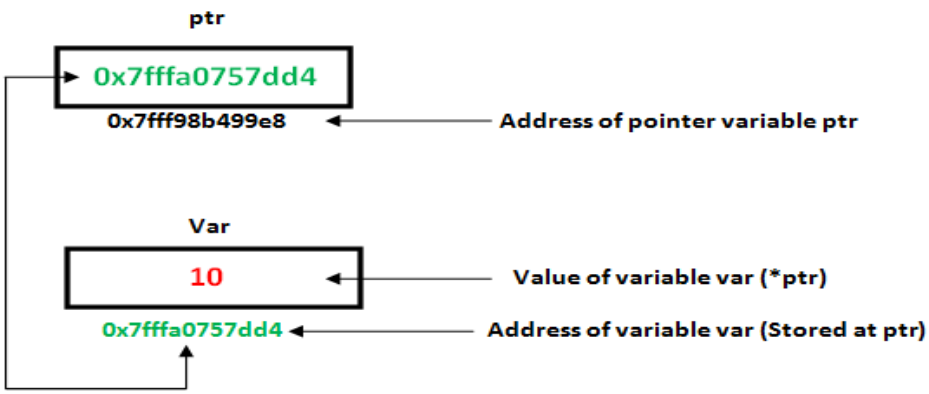
Pointer

- A pointer is a variable which points to the address of another variable of any data type like int ,char ,float etc. Similarly, it have a pointer to structures, where a pointer variable can point to the address of a structure variable.
- We have already learned that a pointer is a variable which points to the address of another variable of any data type like int , char , float etc.
- Similarly, we can have a pointer to structures, where a pointer variable can point to the address of a structure variable.

Syntax:

datatype *var_name;

int *ptr;



- Pointers in C language is a variable that stores/points the address of another variable. A Pointer in C is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.
- Pointers can be used with array and string to access elements more efficiently. We can create function pointers to invoke a function dynamically. Types are null pointer, wild pointer and void pointer.

Application of Pointer:

- To pass arguments by reference
- To return multiple values
- Dynamic memory allocation
- To do system level programming where memory addresses are useful

Video Content / Details of website for further learning (if any):

https://nptel.ac.in/courses/106104128/

https://www.youtube.com/watch?v=6vT1EoPYpTQ

Important Books/Journals for further learning including the page nos.:

E.Horowitz,S.Sahni Susan,Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008- **Page Nos:9-15**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-4

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Dynamic Memory Allocation

Introduction :

- The process of allocating memory at runtime is known as dynamic memory allocation

Prerequisite knowledge for Complete understanding and learning of Topic:

- Memory Management Functions
- Pointers

Detailed content of the Lecture:

Dynamic Memory Allocation

- The process of allocating memory at runtime is known as dynamic memory allocation.
- Library routines known as "memory management functions" are used for allocating and freeing memory during execution of a program.
- These functions are defined in stdlib.h.

Video Content / Details of website for further learning (if any):

https://nptel.ac.in/courses/106105171/

https://www.youtube.com/watch?v=RdY5jiIkCjE

Important Books/Journals for further learning including the page nos.:

E.Horowitz,S.Sahni Susan , Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008- **Page Nos:16-18**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-5

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Functions and Recursion Function

Introduction :

- Recursion is an approach in which a function calls itself with an argument. Upon reaching a termination condition, the control returns to the calling function

Prerequisite knowledge for Complete understanding and learning of Topic:

- Pre-defined function or built-in or intrinsic function
- User defined function

Detailed content of the Lecture:

- These functions are defined in stdlib.h.

Function	Description
malloc()	allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space
calloc()	allocates space for an array of elements, initialize them to zero and then returns a void pointer to the memory
Free	releases previously allocated memory
Realloc	modify the size of previously allocated space

```
Func(int array_size)
 {
double k, a[100], *b, *c;
b = (double *) malloc(array_size * sizeof(double));
c = new double[array_size];
}
```

• The size of the problem often can not be determined at “compile time”.

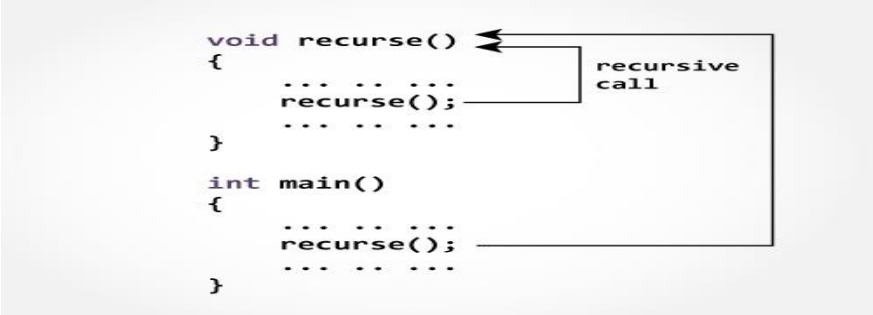
• Dynamic memory allocation is to allocate memory at “run time”.

• Dynamically allocated memory must be referred to by pointers.

Recursion function

Recursion is an approach in which a function calls itself with an argument. Upon reaching a termination condition, the control returns to the calling function. Ex: Factorial

SYNTAX



Advantages

- Reduce unnecessary calling of function.
- Through Recursion one can slove problems in easy way while its iterative solution is very big and complex.

Disdvantages

- Recursive solution is always logical and it is very difficult to trace.(debug and understand).
- In recursive we must have an if statement somewhere to force the function to return without the recursive call being executed, otherwise the function will never return.
- Recursion takes a lot of stack space, usually not considerable when the program is small and running on a PC.
- Recursion uses more processor time.

Video Content / Details of website for further learning (if any):
https://nptel.ac.in/courses/106105171/

https://www.youtube.com/watch?v=RdY5jiIkCjE

Important Books/Journals for further learning including the page nos.:
E.Horowitz,S.Sahni Susan , Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008- **page nos:16-18**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-6

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Linked List: Definition, Types of List, Singly Linked List operations

Introduction :

- A linked list is a non-sequential collection of data items.
- It is a dynamic data structure. For every data item in a linked list, there is an associated pointer that would give the memory location of the next data item in the linked list.
- The data items in the linked list are not in consecutive memory locations.
- It may be anywhere, but the accessing of these data items is easier as each data item contains the address of the next data item.
- A linked list is a non-sequential collection of data items.
- A linked list allocates space for each element separately in its own block of memory called a "node".
- Each node contains two fields; a "data" field to store whatever element, and a "next" field which is a pointer used to link to the next node.

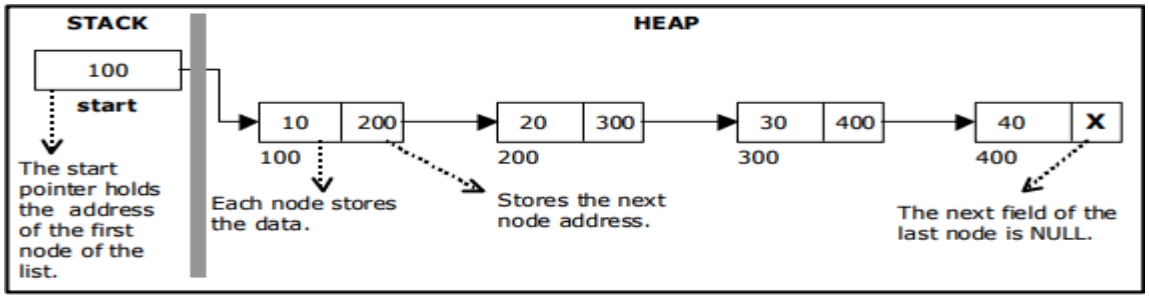
Prerequisite knowledge for Complete understanding and learning of Topic:

- Pointer
- Node
- Memory address
- Linear data structure

Detailed content of the Lecture:

Single Linked list

- It is a linked list ,in which each node contains only one link field pointing to the next node in the list.
- Each node is allocated in the heap using malloc(), so the node memory continues to exist until it is explicitly de-allocated using free(). The front of the list is a pointer to the “start” node.

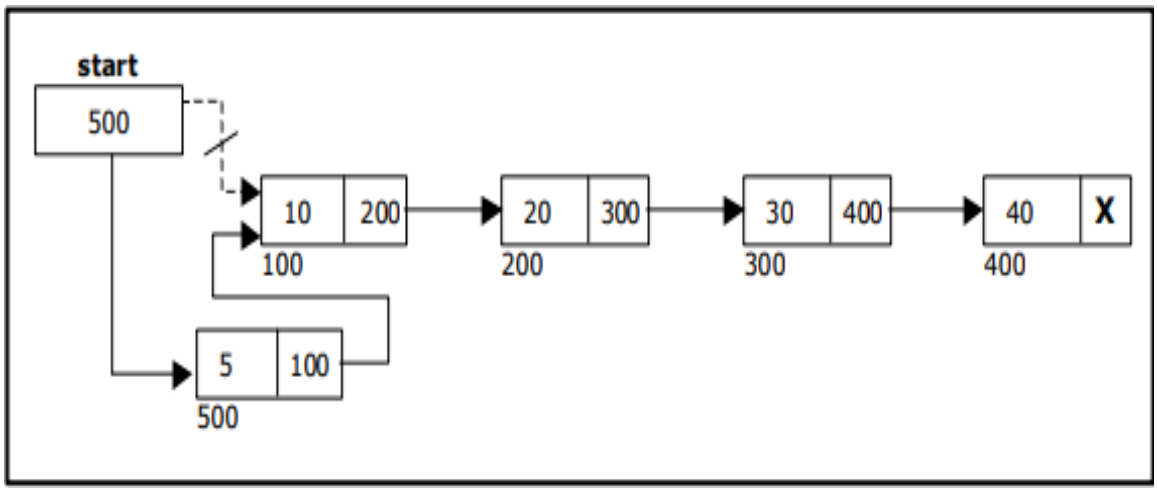


- The beginning of the linked list is stored in a "start" pointer which points to the first node. The first node contains a pointer to the second node.
- The last node in the list has its next field set to NULL to mark the end of the list.

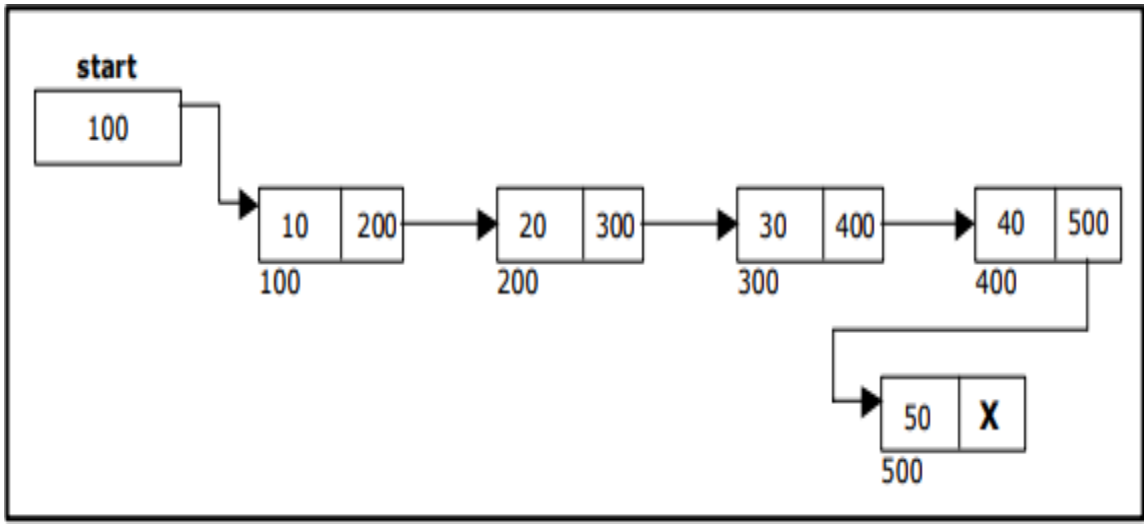
Insertion of a Node:

- One of the most primitive operations that can be done in a singly linked list is the insertion of a node.
- Memory is to be allocated for the new node (in a similar way that is done while creating a list) before reading the data.

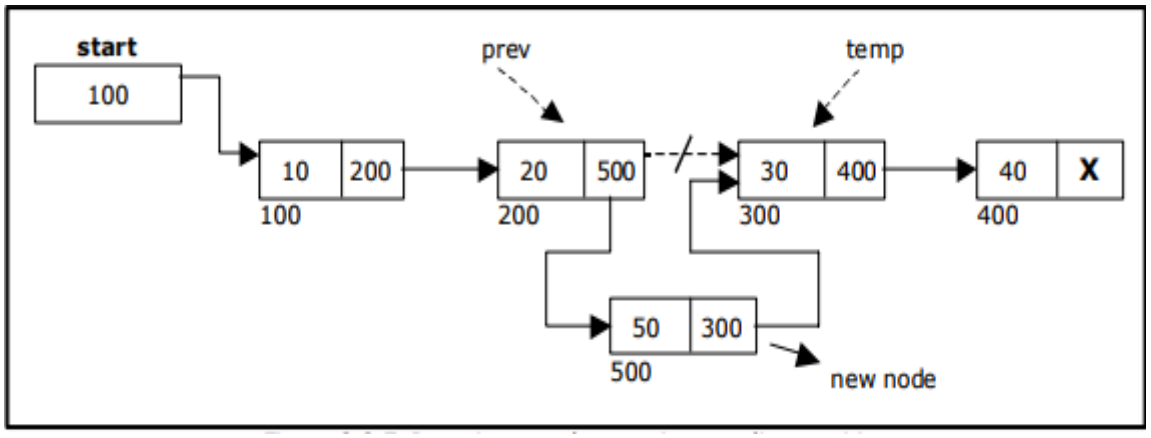
1. **Inserting a node at the beginning**



2. .**Inserting a node at the end**.



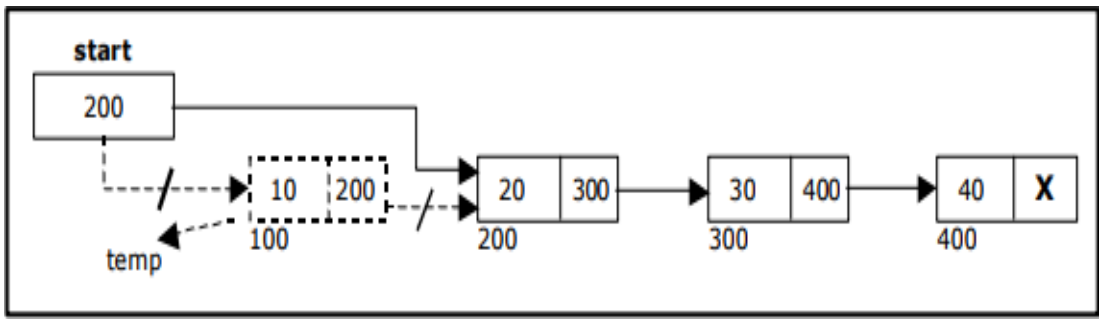
3. **Inserting a node at intermediate position.**



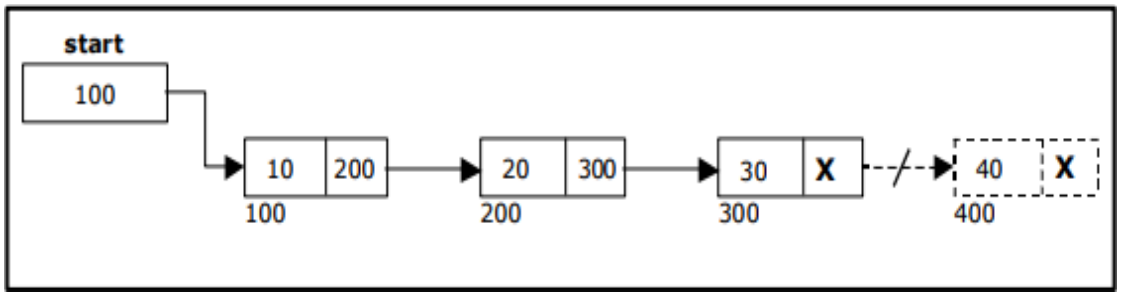
Deletion of a node

- Another primitive operation that can be done in a singly linked list is the deletion of a node.
- Memory is to be released for the node to be deleted.

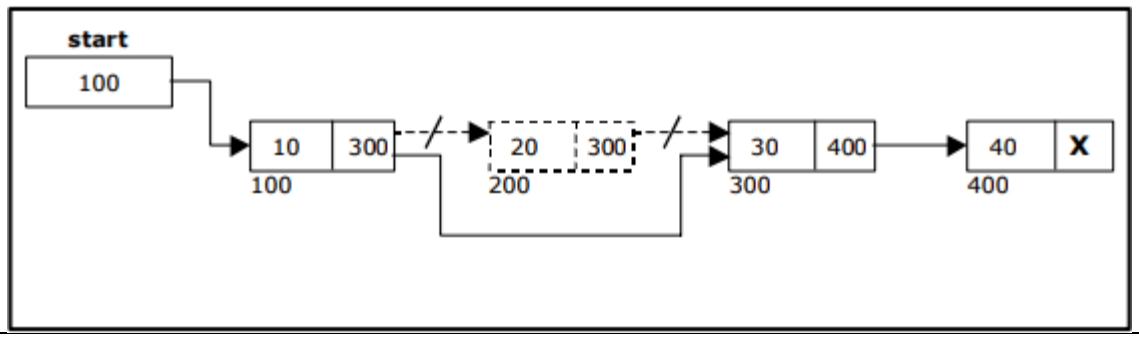
1. **Deleting a node at the beginning.**



2. **Deleting a node at the end.**



3. **Deleting a node at intermediate position.**



Video Content / Details of website for further learning (if any):

https://youtu.be/PGWZUgzDMYI

https://nptel.ac.in/courses/106105085/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012- **page nos:57-65**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-7

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Doubly Linked list operation

Introduction :

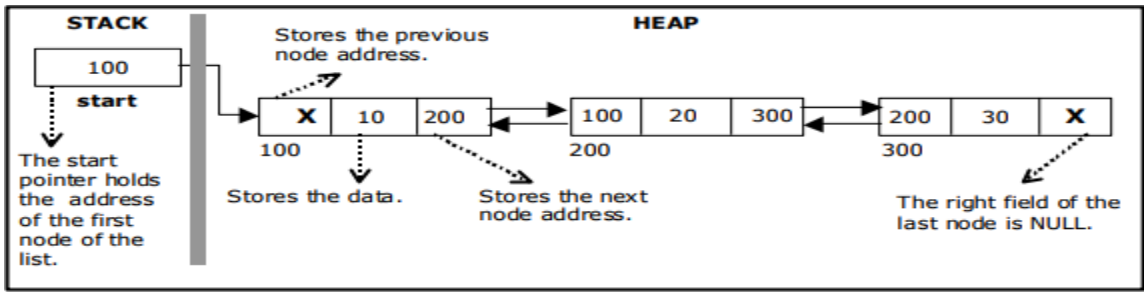
- A double linked list is a two-way list in which all nodes will have two links.
- The left link points to the predecessor node
- The right link points to the successor node. The data field stores the required data.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Linked list
- Node
- Memory allocation
- Pointer

Detailed content of the Lecture:

- It is a list in which each node has three fields namely data field, forward link and backward link



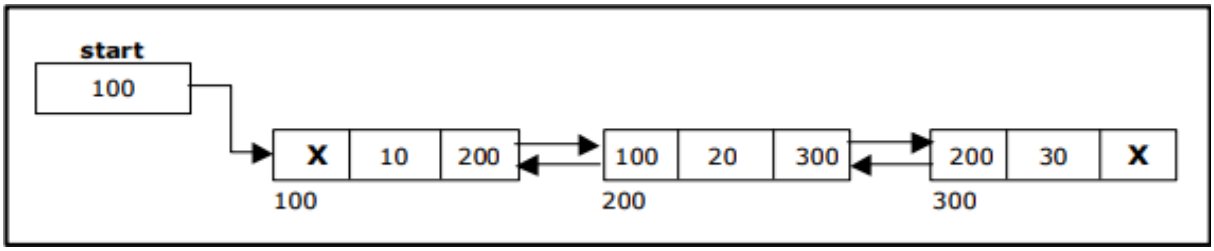
- The beginning of the double linked list is stored in a "start" pointer which points to the first node. The first node's left link and last node's right link is set to NULL.

The basic operations in a double linked list are:

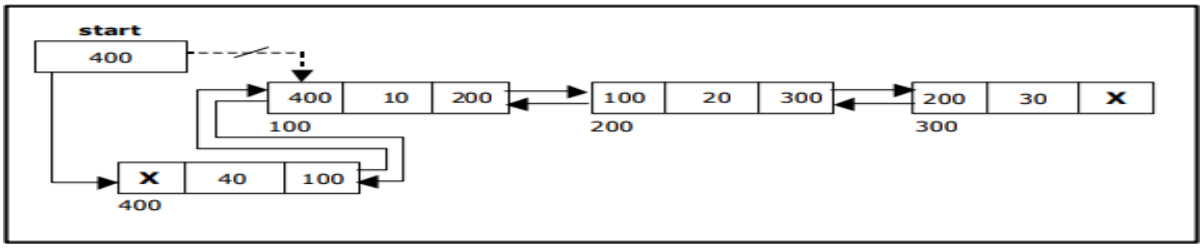
- Creation.
- Insertion.
- Deletion.
- Traversing.

Creating a node for Double Linked List

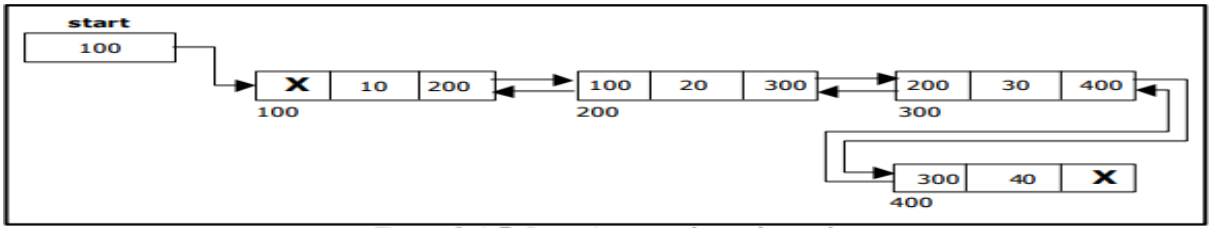
Double Linked List with 3 nodes



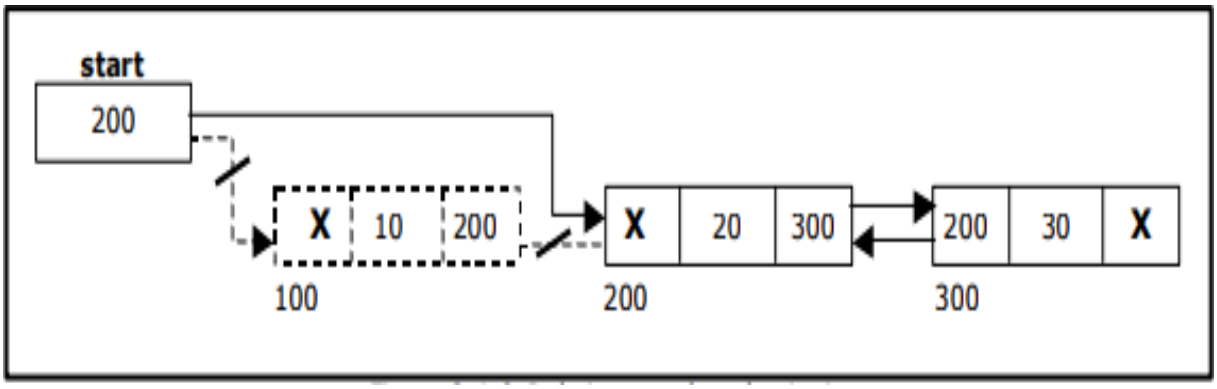
Inserting a node at the beginning



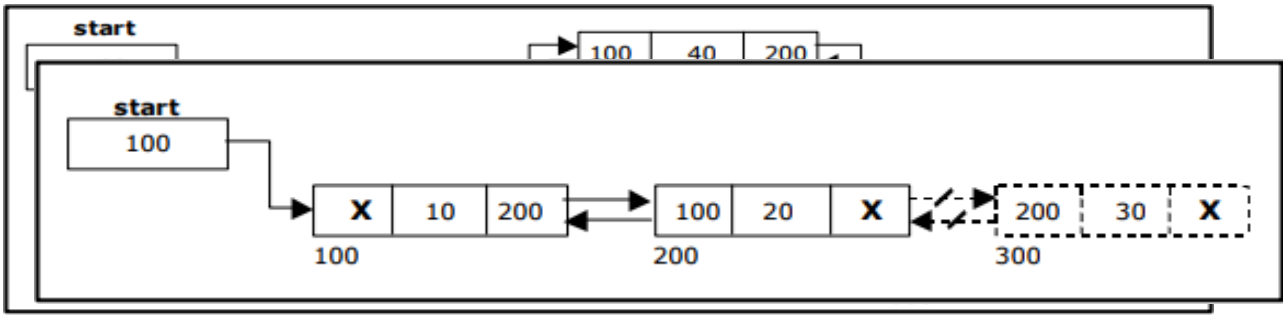
Inserting a node at the end



Inserting a node at an intermediate position:

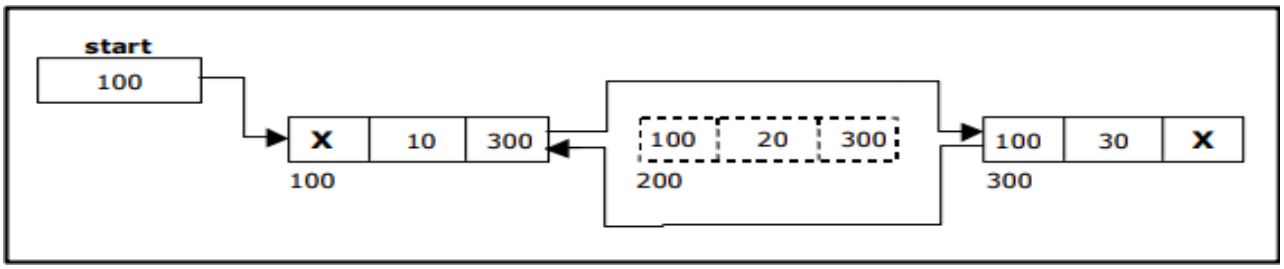


Deleting a node at the beginning:



Deleting a node at the end:

Deleting a node at Intermediate position:



Traversal and displaying a list (Left to Right):

- To display the information, you have to traverse the list, node by node from the first node, until the end of the list is reached. The function traverse_left_right() is used for traversing and displaying the information stored in the list from left to right.

Traversal and displaying a list (Right to Left):

To display the information from right to left, you have to traverse the list, node by node from the first node, until the end of the list is reached. The function traverse_right_left() is used for traversing and displaying the information stored in the list from right to left.

Video Content / Details of website for further learning (if any):

https://youtu.be/PGWZUgzDMYI

https://nptel.ac.in/courses/106105085/

Important Books/Journals for further learning including the page nos.:

Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012- **page nos:65-67**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L-8

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Circular linked list operation

Introduction :

- It is just a single linked list in which the link field of the last node points back to the address of the first node.
- A circular linked list has no beginning and no end. It is necessary to establish a special pointer called start pointer always pointing to the first node of the list.
- In circular linked list no null pointers are used, hence all pointers contain valid address.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Linked list(Single, Double)
- Node
- Memory allocation
- Pointer

Detailed content of the Lecture:

Circular linked list

- Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end.

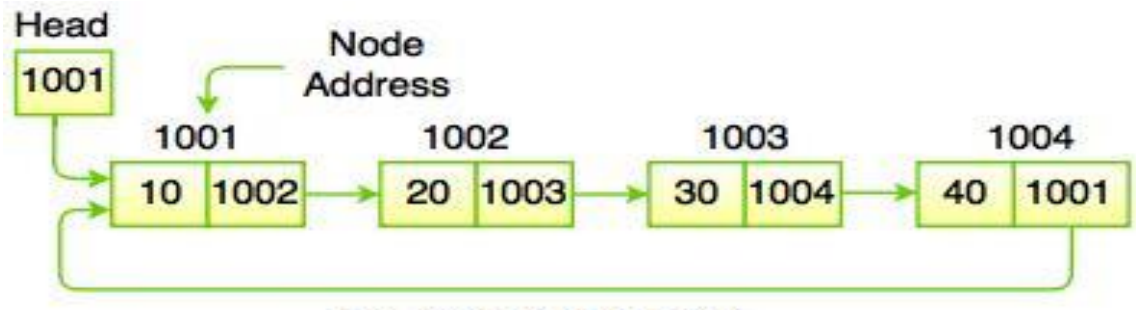
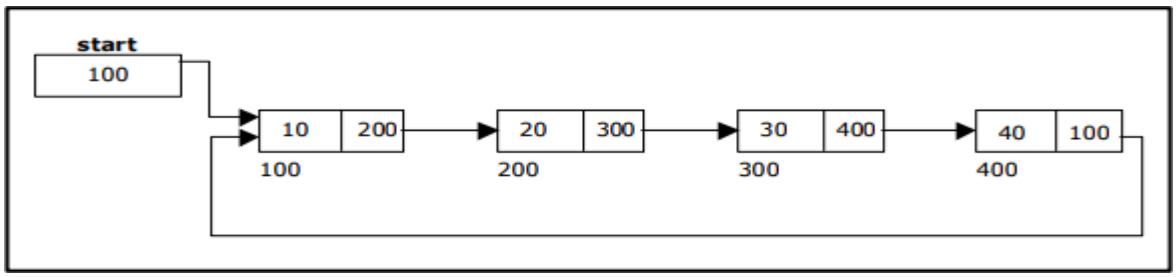


Fig. Circular Linked List

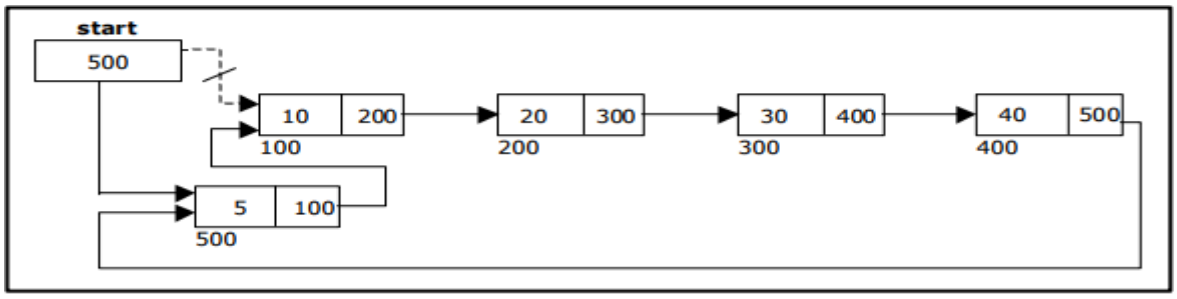
Basic operations in a circular double linked list are:

- Creation.
- Insertion.
- Deletion.
- Traversing.

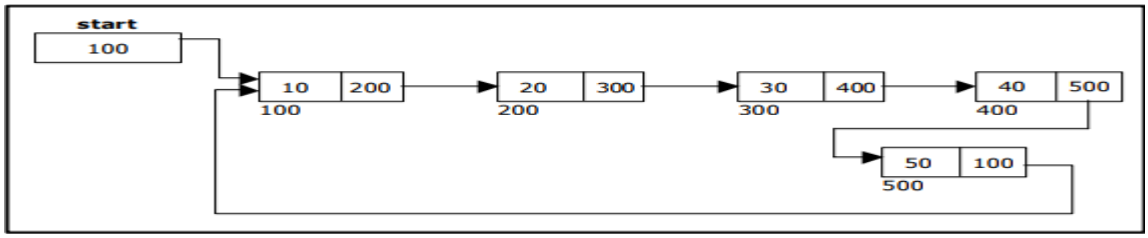
Creating a circular single Linked List



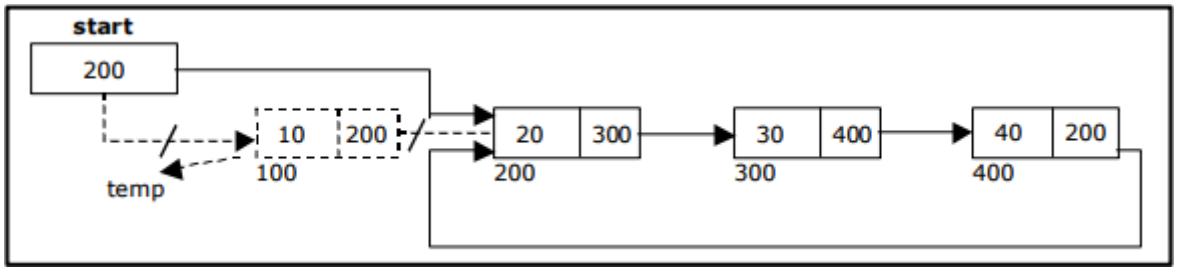
Inserting a node at the beginning



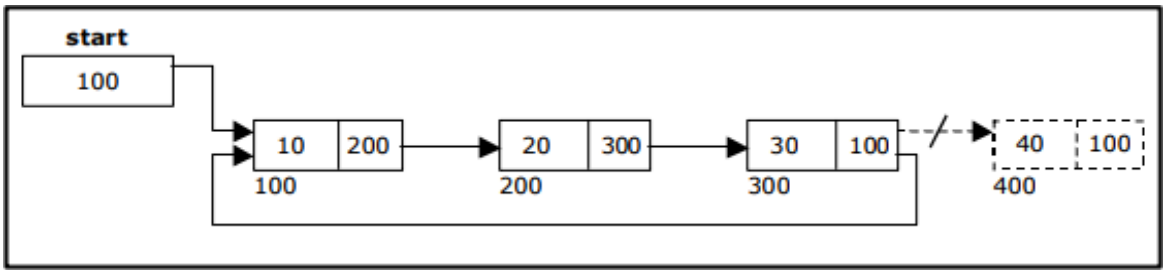
Inserting a node at the end:



Deleting a node at the beginning:



Deleting a node at the end:



Video Content / Details of website for further learning (if any):

https://youtu.be/PGWZUgzDMYI

https://nptel.ac.in/courses/106105085

Important Books/Journals for further learning including the page nos.:

R. F. Gilberg B. A. Forouzan Data Structures2nd Edition, Thomson India2005, **page nos: 22-24**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-9

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : I [INTRODUCTION AND LIST] **Date of Lecture:**

Topic of Lecture: Applications of linked list

Introduction :

- Implementation of stacks and queues.
- Implementation of graphs : Adjacency list representation of graphs is most popular which is uses linked list to store adjacent vertices

Prerequisite knowledge for Complete understanding and learning of Topic:

- Singly Linked List
- Doubly Linked List
- Circular Linked List

Detailed content of the Lecture:

- Radix sort
- Multi List
- Polynomial list

Radix sort

- Radix sort is one of the sorting algorithms used to sort a list of integer numbers in order.
- In radix sort algorithm, a list of integer numbers will be sorted based on the digits of individual numbers.
- For example, if the largest number is a 3 digit number then that list is sorted with 3 passes.

The Radix sort algorithm is performed using the following steps

Step 1 - Define 10 queues each representing a bucket for each digit from 0 to 9.

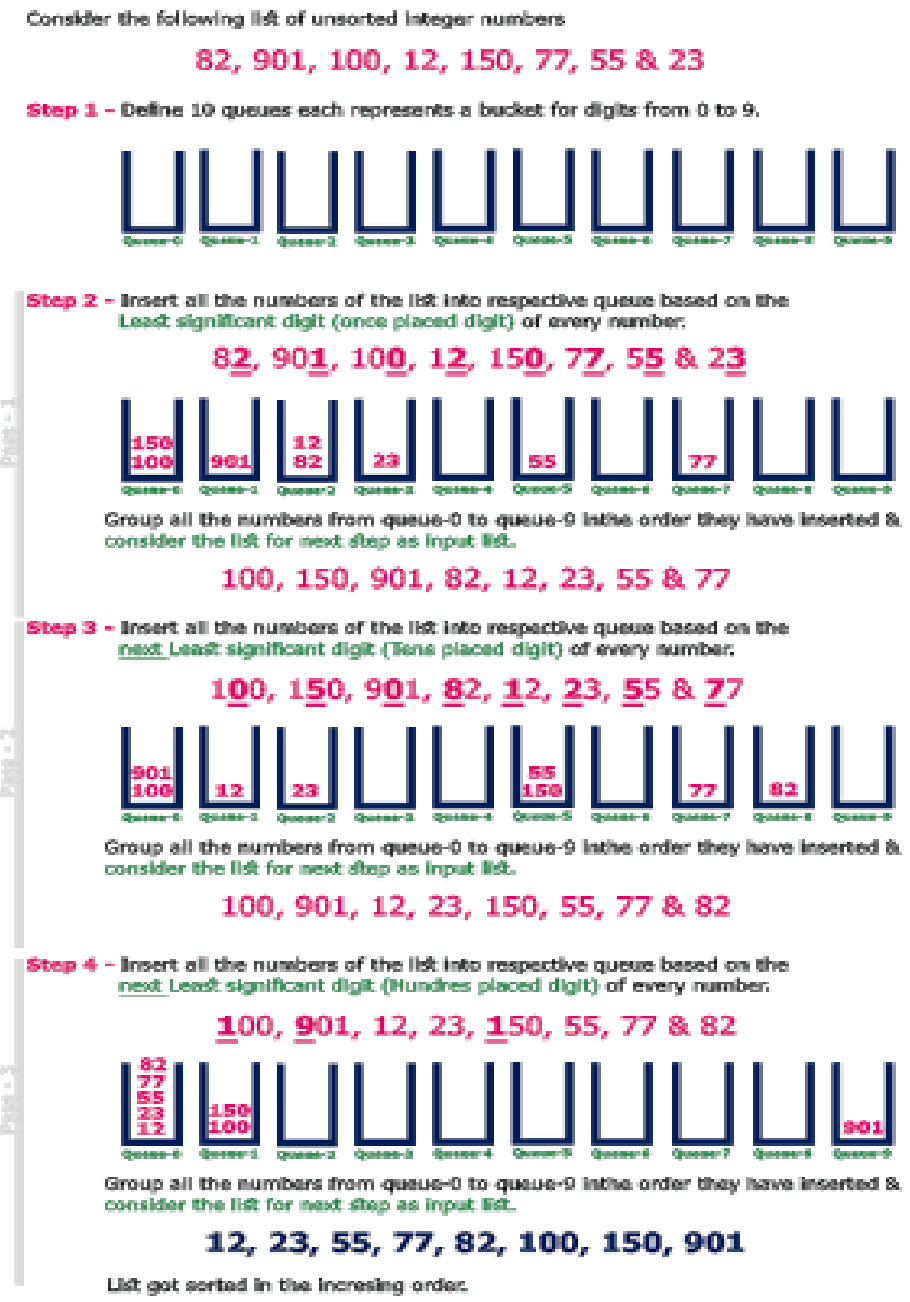
Step 2 - Consider the least significant digit of each number in the list which is to be sorted.

Step 3 - Insert each number into their respective queue based on the least significant digit.

Step 4 - Group all the numbers from queue 0 to queue 9 in the order they have inserted into their respective queues.

Step 5 - Repeat from step 3 based on the next least significant digit.

Step 6 - Repeat from step 2 until all the numbers are grouped based on the most significant digit.



Video Content / Details of website for further learning (if any):

www.btechsmartclass.com › data_structures › radix-sort

Important Books/Journals for further learning including the page nos.:

R. F. Gilberg B. A. Forouzan Data Structures2nd Edition, Thomson India2005, page nos: **24-26**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-10

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Stack: Stack-Definitions & Concepts, Array implementation of Stack

Introduction :

- Stack is an Abstract data structure (ADT) works on the principle Last In First Out (LIFO).
- The last element add to the stack is the first element to be delete. Insertion and deletion can be takes place at one end called TOP. It looks like one side closed tube.
- Stack implemented using array stores only a fixed number of data values

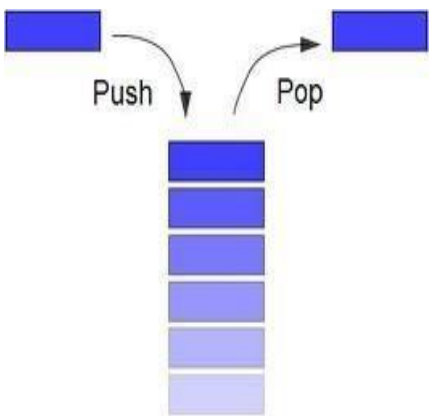
Prerequisite knowledge for Complete understanding and learning of Topic:

- Abstract data Type
- Arrays
- Linked List

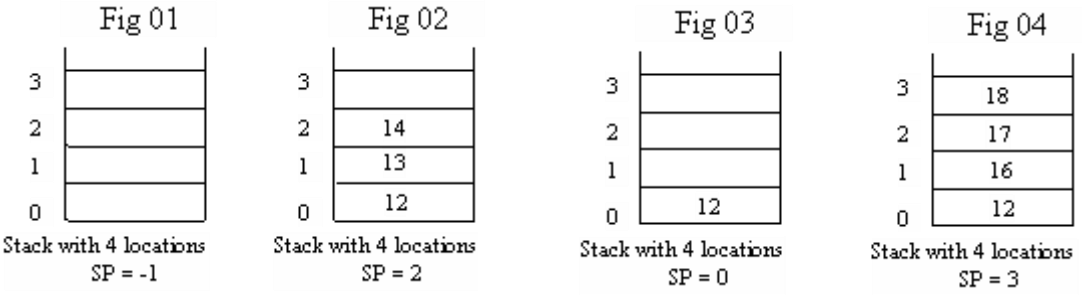
Detailed content of the Lecture:

- A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack.

  A stack is a limited access data structure - elements can be added and removed from the stack only at the top. Push adds an item to the top of the stack, pop removes the item from the top.



- The add operation of the stack is called push operation
- The delete operation is called as pop operation.
- Push operation on a full stack causes stack overflow.
- Pop operation on an empty stack causes stack underflow.
- SP is a pointer, which is used to access the top element of the stack.
- If you push elements that are added at the top of the stack
- In the same way when we pop the elements, the element at the top of the stack is deleted.



There are two operations applied on stack they are

- Push
- pop.

While performing push & pop operations the following test must be conducted on the stack.

- Stack is empty or not
- Stack is full or not

Push:

- Push operation is used to add new elements in to the stack. At the time of addition first check the stack is full or not. If the stack is full it generates an error message "stack overflow".

Pop:

- Pop operation is used to delete elements from the stack. At the time of deletion first check the stack is empty or not. If the stack is empty it generates an error message "stack underflow".

Array implementation of Stack Operations

Push(Value) - Inserting Value into The Stack

- In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at **top** position. Push function takes one integer value as parameter and inserts that value into the stack. It can use the following steps to push an element on to the stack...
- **Step 1 -** Check whether **stack** is **FULL**. (**Top == SIZE-1**)
- **Step 2 -** If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

Pop() - Delete A Value From The Stack

- In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from **top** position. Pop function does not take any value as parameter. It can use the following steps to pop an element from the stack.

Step 1 - Check whether **stack** is **EMPTY**. (**Top == -1**)

Step 2 - If it is **EMPTY**, then display **"Stack is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then delete **stack[top]** and decrement **top** value by one (**top--**).

Video Content / Details of website for further learning (if any):

https://www.wiziq.com/tutorials/data-structure

https://nptel.ac.in/courses/106/106/106106133/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **Page Nos:** 78-79

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L-11

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Linked list implementation of Stack, Operations on Stacks

Introduction :

- Stack implemented using linked list, the nodes are maintained non-contiguously in the memory. Each node contains a pointer to its immediate successor node in the stack

Prerequisite knowledge for Complete understanding and learning of Topic:

- Stack
- Nodes
- Array
- Linked list

Detailed content of the Lecture:

Linked list implementation of Stack Operations

Push(Value) - Inserting an Element into The Stack

- It can use the following steps to insert a new node into the stack.
- **Step 1 -** Create a **newNode** with given value.
- **Step 2 -** Check whether stack is **Empty** (**top** == **NULL**)
- **Step 3 -** If it is **Empty**, then set **newNode** → **next** = **NULL**.
- **Step 4 -** If it is **Not Empty**, then set **newNode** → **next** = **top**.
- **Step 5 -** Finally, set **top** = **newNode**.

Pop() – Deleting an Element From A Stack

- It can use the following steps to delete a node from the stack...
- **Step 1 -** Check whether **stack** is **Empty** (**top == NULL**).
- **Step 2 -** If it is **Empty**, then display **"Stack is Empty!!! Deletion is not possible!!!"** and terminate the function
- **Step 3 -** If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.
- **Step 4 -** Then set '**top** = **top** → **next**'.
- **Step 5 -** Finally, delete '**temp**'. (**free(temp)**).

Video Content / Details of Itbsite for further learning (if any):

https://www.youtube.com/watch?v=sFVxsglODoo

https://nptel.ac.in/courses/106/106/106106133/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 79-86

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-12

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Applications of Stacks, Polish Expression, Reverse Polish Expression

Introduction :

- Stacks can be used to check parenthesis matching in an expression.
- Stacks can be used for Conversion from one form of expression to another.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Stack operation
- Expression
- Recursion function
- Stack

Detailed content of the Lecture:

- Polish Expression
- Reverse Polish Expression
- Recursion
- Tower of Hanoi

Recursion

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.
- Examples of such problems are Towers of Hanoi

Tower of Hanoi :

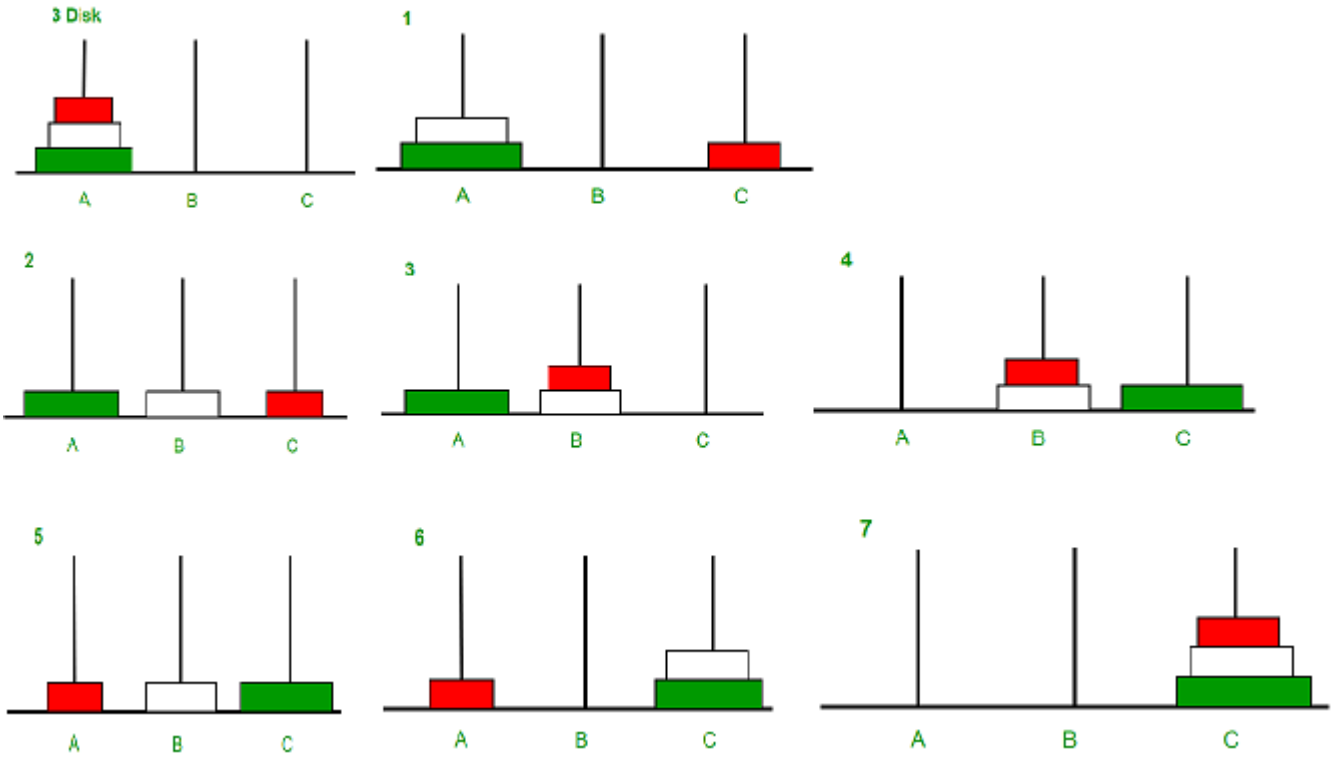
- It is a mathematical puzzle where we have three rods and n disks.
- The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

  1) Only one disk can be moved at a time.

  2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack

- i.e. a disk can only be moved if it is the uppermost disk on a stack.

  3) No disk may be placed on top of a smaller disk.



Polish Expression:

Expressions are divided into Three types

- Infix Expression
- Postfix Expression
- Prefix Expression

Infix Expression

In infix expression, operator is used in between the operands.

Postfix Expression

In postfix expression, operator is used after operands that the "Operator follows the Operands".

Prefix Expression

In prefix expression, operator is used before operands that the "Operands follows the Operator

Conversion of infix expression to postfix expression

- Scan the infix expression from left to right.
- If the scanned symbol is left parenthesis, push it onto the stack.
- If the scanned symbol is an operand, then place directly in the postfix expression (output).

- If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.
- If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

Symbol	Postfix String	Stack
A	A	
+	A	+
B	A B	+
*	A B	+ *
C	A B C	-
-	A B C * +	-
D	A B C * + D	-
/	A B C * + D	- /
E	A B C * + D E	- /
*	A B C * + D E /	- *
H	A B C * + D E / H	- *
End of string	A B C * + D E / H * -	The input is now empty. Pop the output symbols from the stack until it is empty

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=sFVxsglODoo

https://nptel.ac.in/courses/106/106/106106133/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **Page Nos:** 87-92

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

L-13

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Reverse Polish Expression and their Compilation,

Introduction :

- When a function calls itself, it's called Recursion.
- Tower of Hanoi is a mathematical puzzle where we have three rods and n disks.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Infix Expression
- Prefix Expression
- Postfix Expression

Detailed content of the Lecture:

Polish Notation:

There are Three Types of Polish Notation.

1. Prefix = +ab
2. Infix = a+b
3. Postfix =ab+

Solve the Following Expression For Both Prefix and Postfix:

(A+B) / C * D – E

Prefix:

{+ A B } / C * D - E

= { / * A B C } * D - E

= { * / + A B C D } - E

={ - * / + A B C D E }

Postfix : = { A B + } / C * D - E

= { A B + C / } * D - E

= { A B + C / D * } - E

= { A B + C / D * E - }

Video Content / Details of Itbsite for further learning (if any):
https://en.wikipedia.org/wiki/Reverse_Polish_notation
Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos: 92-94**

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

LECTURE HANDOUTS | **L-14**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Queue: Representation of Queue

Introduction :

- A queue is a data structure that is best described as "first in, first out".
- A queue is another special kind of list, where items are inserted at one end called the rear and deleted at the other end called the front.
- A real world example of a queue is people waiting in line at the bank. As each person enters the bank, "en-queued" at the back of the line.
- When a teller becomes available, they are "dequeued" at the front of the line.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Data structure
- Indices
- Pointers

Detailed content of the Lecture:

- Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other end called as FRONT.
- Front points to the beginning of the queue and Rear points to the end of the queue.
- Queue follows the FIFO (First - In - First Out) structure.

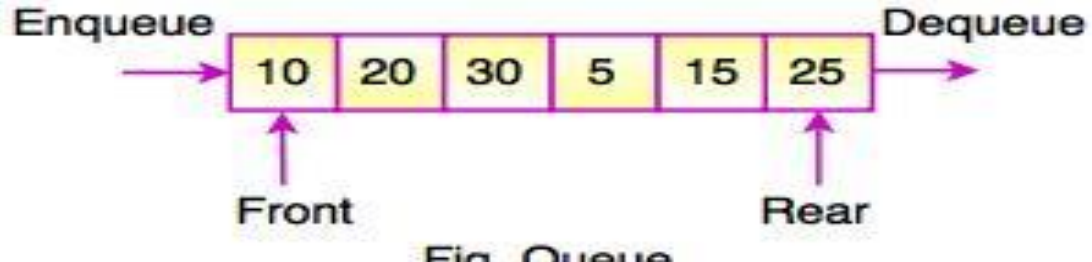


Fig. Queue

- **Enqueue** means to insert an item into the back of the **queue.**

  Step 1 − Check if the queue is empty.

  Step 2 − If the queue is empty, produce underflow error and exit.

  Step 3 − If the queue is not empty, access the data where front is pointing.

  Step 4 − Increment front pointer to point to the next available data element and success.

- **Dequeue** means removing the front item. The picture demonstrates the FIFO access. The difference between stacks and queues is in removing.

  Step 1 − Check if the queue is full.

  Step 2 − If the queue is full, produce overflow error and exit.

  Step 3 − If the queue is not full, increment rear pointer to point the next empty space.

  Step 4 − Add data element to the queue location, where the rear is pointing and return success.

Operations on Queue

Operations	Description
enqueue()	This function defines the operation for adding an element into queue.
dequeue()	This function defines the operation for removing an element from queue.
init()	This function is used for initializing the queue.
Front	Front is used to get the front data item from a queue.
Rear	Rear is used to get the last item from a queue.

Video Content / Details of website for further learning (if any):

https://nptel.ac.in/courses/106/106/106106127/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **Page Nos:** 95,96

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-15**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Array implementation of Queue Operations

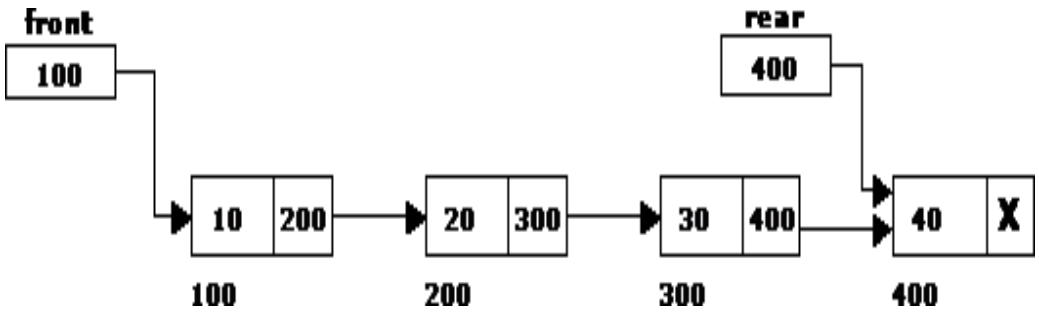
Introduction :

- We can represent a queue as a linked list.
- In a queue data is deleted from the front end and inserted at the rear end.
- We can perform similar operations on the two ends of a list.
- We use two pointers front and rear for our linked queue implementation.

Prerequisite knowledge for Complete understanding and learning of Topic:

- linked list
- pointers

Detailed content of the Lecture:



- A queue data structure can be implemented using a linked list data structure.
- The queue which is implemented using a linked list can work for an unlimited number of values.
- That means, queue using linked list can work for the variable size of data (No need to fix the size at the beginning of the implementation).
- The Queue implemented using linked list can organize as many data values as we want.

Queue Operations using Array

- Queue data structure using array can be implemented as follows.
- Before implement actual operations, first follow the below steps to create an empty queue.

  Step 1 - Include all the **header files** which are used in the program and define a constant '**SIZE**' with specific value.

  Step 2 - Declare all the **user defined functions** which are used in queue implementation.

  Step 3 - Create a one dimensional array with above defined SIZE (**int queue[SIZE]**)

  Step 4 - Define two integer variables '**front**' and '**rear**' and initialize both with '**-1**'. (**int front = -1, rear = -1**)

  Step 5 - Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

enQueue(value) - Inserting value into the queue

- In a queue data structure, enQueue() is a function used to insert a new element into the queue.
- In a queue, the new element is always inserted at **rear** position.
- The enQueue() function takes one integer value as a parameter and inserts that value into the queue.

Steps to insert an element into the queue.

Step 1 - Check whether **queue** is **FULL**. (**rear == SIZE-1**)

Step 2 - If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

deQueue() - Deleting a value from the Queue

- In a queue data structure, deQueue() is a function used to delete an element from the queue.
- In a queue, the element is always deleted from **front** position.
- The deQueue() function does not take any value as parameter.

Steps to delete an element from the queue.

Step 1 - Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check whether both **front** and **rear** are equal (**front** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

display() - Displays the elements of a Queue

Steps to display the elements of a queue.

Step 1 - Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front+1**'.

Step 4 - Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i**' value reaches to **rear** (**i** <= **rear**)

Video Content / Details of website for further learning (if any):
https://nptel.ac.in/courses/106/106/106106127/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **Page Nos:** 96,97

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L-16**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Linked list implementation of Queue Operations

Introduction :

- The implementation of queue data structure using array is very simple.
- Just define a one dimensional array of specific size and insert or delete the values into that array by using FIFO (First In First Out) principle with the help of variables '**front**' and '**rear**. Initially both '**front**' and '**rear**' are set to -1.
- Whenever, insert a new value into the queue, increment '**rear**' value by one and then insert at that position.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Queue
- Pointers
- Front
- Rear

Detailed content of the Lecture:

Operations

Steps to implement queue using linked list.

Step 1 - Include all the **header files** which are used in the program. And declare all the **user defined functions.**

Step 2 - Define a '**Node**' structure with two members **data** and **next.**

Step 3 - Define two **Node** pointers '**front**' and '**rear**' and set both to **NULL**.

Step 4 - Implement the **main** method by displaying Menu of list of operations and make suitable function calls in the **main** method to perform user selected operation.

enQueue(value) - Inserting an element into the Queue

Steps to insert a new node into the queue.

Step 1 - Create a **newNode** with given value and set '**newNode → next**' to **NULL**.

Step 2 - Check whether queue is **Empty** (**rear** == **NULL**)

Step 3 - If it is **Empty** then, set **front** = **newNode** and **rear** = **newNode**.

Step 4 - If it is **Not Empty** then, set **rear → next** = **newNode** and **rear** = **newNode**.

deQueue() - Deleting an Element from Queue

Steps to delete a node from the queue.

Step 1 - Check whether **queue** is **Empty** (**front == NULL**).

Step 2 - If it is **Empty**, then display **"Queue is Empty!!! Deletion is not possible!!!"** and terminate from the function

Step 3 - If it is **Not Empty** then, define a Node pointer '**temp**' and set it to '**front**'.

Step 4 - Then set '**front** = **front → next**' and delete '**temp**' (**free(temp)**).

display() - Displaying the elements of Queue

Steps to display the elements (nodes) of a queue.

Step 1 - Check whether queue is **Empty** (**front** == **NULL**).

Step 2 - If it is **Empty** then, display **'Queue is Empty!!!'** and terminate the function.

Step 3 - If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **front**.

Step 4 - Display '**temp → data** --->' and move it to the next node. Repeat the same until '**temp**' reaches to '**rear**' (**temp → next** != **NULL**).

Step 5 - Finally! Display '**temp → data** ---> **NULL**'.

Video Content / Details of website for further learning (if any):

https://nptel.ac.in/courses/106/106/106106127/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **Page Nos:** 99,100

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L-17**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Circular Queue, Priority Queue, Array representation of Priority Queue

Introduction :

- A circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle

Prerequisite knowledge for Complete understanding and learning of Topic:

- linear data structure
- Queue

Detailed content of the Lecture:

Implementation of Circular Queue

To implement a circular queue data structure using an array, we first perform the following steps before we implement actual operations.

Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.

Step 2 - Declare all user defined functions used in circular queue implementation.

Step 3 - Create a one dimensional array with above defined SIZE (int cQueue[SIZE]**)**

Step 4 - Define two integer variables 'front' and 'rear' and initialize both with '-1'**.** **(**int front = -1,rear = -1**)**

Step 5 - Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

enQueue(value) - Inserting value into the Circular Queue

- In a circular queue, enQueue() is a function which is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at **rear** position.
- The enQueue() function takes one integer value as parameter and inserts that value into the circular queue. We can use the following steps to insert an element into the circular queue..

Step 1 - Check whether queue is FULL**. (**(rear == SIZE-1 && front == 0) || (front == rear+1)**)**
Step 2 - If it is FULL**,** then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.
Step 3 - If it is NOT FULL, then check rear == SIZE - 1 && front != 0 if it is TRUE**,** then set rear = -1**.**
Step 4 - Increment rear value by one **(**rear++**),** set queue[rear] **=** value and check 'front == -1' if it is TRUE**,** then set front = 0**.**

deQueue() - Deleting a value from the Circular Queue

- In a circular queue, deQueue() is a function used to delete an element from the circular queue.
- In a circular queue, the element is always deleted from **front** position.
- The deQueue() function doesn't take any value as a parameter. We can use the following steps to delete an element from the circular queue.

Step 1 - Check whether queue is EMPTY**. (**front == -1 && rear == -1**)**

Step 2 - If it is EMPTY**,** then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.

Step 3 - If it is NOT EMPTY**,** then display queue[front] as deleted element and increment the front value by one **(**front ++). Then check whether front == SIZE**,** if it is TRUE**,** then set front = 0**.** Then check whether both front - 1 and rear are equal **(**front -1 == rear), if it TRUE**,** then set both front and rear to '-1' **(**front = rear = -1**).**

display() - Displays the elements of a Circular Queue

Steps to display the elements of a circular queue.

Step 1 - Check whether queue is EMPTY**. (**front == -1**)**

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then define an integer variable '**i**' and set 'i = front'.

Step 4 - Check whether 'front <= rear', if it is TRUE**,** then display 'queue[i]' value and increment 'i' value by one (**i++**). Repeat the same until 'i <= rear' becomes FALSE**.**

Step 5 - If 'front <= rear' is FALSE**,** then display 'queue[i]' value and increment '**i**' value by one (**i++**). Repeat the same until'i <= SIZE - 1' becomes FALSE**.**

Step 6 - Set i to 0.

Step 7 - Again display 'cQueue[i]' value and increment **i** value by one (**i++**). Repeat the same until 'i <= rear' becomes FALSE**.**

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=ImSRt7LxQnY

Important Books/Journals for further learning including the page nos.:

R.Kruse,C.L.Tondo and B.Leung,Data structures and Program Design in C, 2nd Edition , Prentice-Hall, 2006,**page nos:** 27-32

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-18**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : II [STACK AND QUEUE] **Date of Lecture:**

Topic of Lecture: Double Ended Queue, Applications of Queue

Introduction :

- Insertion is performed at the end of the queue and deletion is performed based on the FIFO principle.
- This queue implementation may not be suitable for all applications

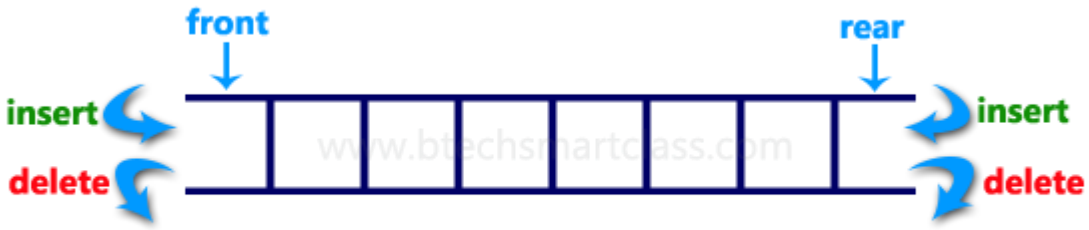
Prerequisite knowledge for Complete understanding and learning of Topic:

- Queue implementation

Detailed content of the Lecture:

Double Ended Queue Datastructure

- Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (**front** and **rear**). Insert at both front and rear positions and can delete from both front and rear positions.

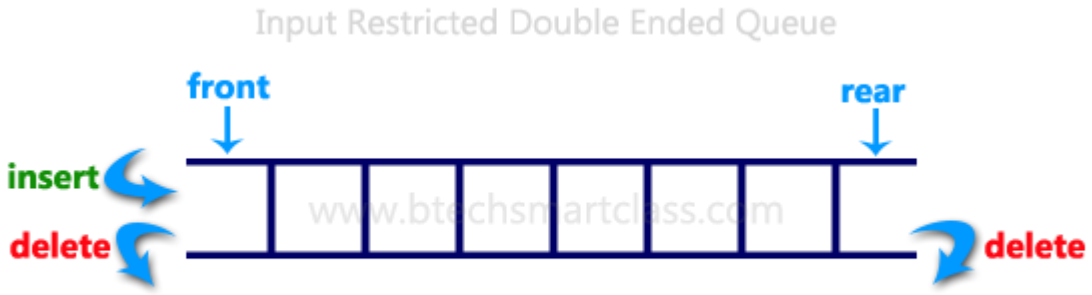


Double Ended Queue can be represented in TWO ways,

- Input Restricted Double Ended Queue
- Output Restricted Double Ended Queue

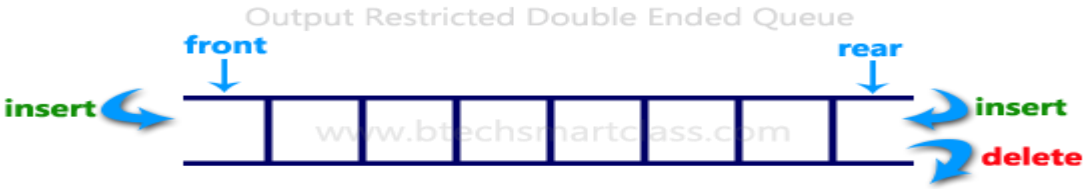
Input Restricted Double Ended Queue

- In input restricted double-ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.



Output Restricted Double Ended Queue

- In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.



APPLICATION OF QUEUE

- Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
- In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.

Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e,

First come first served

There are two types of priority queues are as follows.

1. **Max Priority Queue**
2. **Min Priority Queue**

1. Max Priority Queue

- In a max priority queue, elements are inserted in the order in which they arrive the queue and the maximum value is always removed first from the queue.
- For example, assume that we insert in the order 8, 3, 2 & 5 and they are removed in the order 8, 5, 3, 2.

- The following are the operations performed in a Max priority queue.

1. **isEmpty() - Check whether queue is Empty.**
2. **insert() - Inserts a new value into the queue.**
3. **findMax() - Find maximum value in the queue.**
4. **remove()** - Delete maximum value from the queue.

Max Priority Queue Representations

There are 6 representations of max priority queue.

- Using an Unordered Array (Dynamic Array)
- Using an Unordered Array (Dynamic Array) with the index of the maximum value
- Using an Array (Dynamic Array) in Decreasing Order
- Using an Array (Dynamic Array) in Increasing Order
- Using Linked List in Increasing Order
- Using Unordered Linked List with reference to node with the maximum value

    - **isEmpty()** - If '**front == -1**' queue is Empty. This operation requires **O(1)** time complexity which means constant time complexity.
    - **insert()** - New element is added at the end of the queue. This operation requires **O(1)** time complexity which means constant time complexity.
    - **findMax()** - To find the maximum element in the queue, we need to compare it with all the elements in the queue. This operation requires **O(n)** time complexity.
    - **remove()** - To remove an element from the max priority queue, first we need to find the largest element using **findMax()** which requires **O(n)** time complexity, then that element is deleted with constant time complexity **O(1)**. The remove() operation requires **O(n) + O(1) ≈ O(n)** time complexity.

Min Priority Queue Representations

- Min Priority Queue is similar to max priority queue except for the removal of maximum element first.
- Remove minimum element first in the min-priority queue.

The following operations are performed in Min Priority Queue.

1. **isEmpty()** - Check whether queue is Empty.
2. **insert()** - Inserts a new value into the queue.
3. **findMin()** - Find minimum value in the queue.
4. **remove()** - Delete minimum value from the queue

Video Content / Details of website for further learning (if any):

https://nptel.ac.in/courses/106/106/106106127/

https://nptel.ac.in/courses/106/106/106106127/

Important Books/Journals for further learning including the page nos.:

R.Kruse,C.L.Tondo and B.Leung,Data structures and Program Design in C, 2nd Edition , Prentice-Hall, 2006,**Page Nos:** 33-35

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L-19**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III [TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Trees: Basic terminologies of trees

Introduction :

- Tree is a non-linear data structure which organizes data in a hierarchical structure and this is a recursive definition.
- A tree is a connected graph without any circuits.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Two important topics)

- **Non-linear data Structure**
- **Data**

Detailed content of the Lecture:

- Tree is a hierarchical data structure which stores the information naturally in the form of hierarchy style.
- Tree is one of the most powerful and advanced data structures.
- It is a non-linear data structure compared to arrays, linked lists, stack and queue.
- It represents the nodes connected by edges.

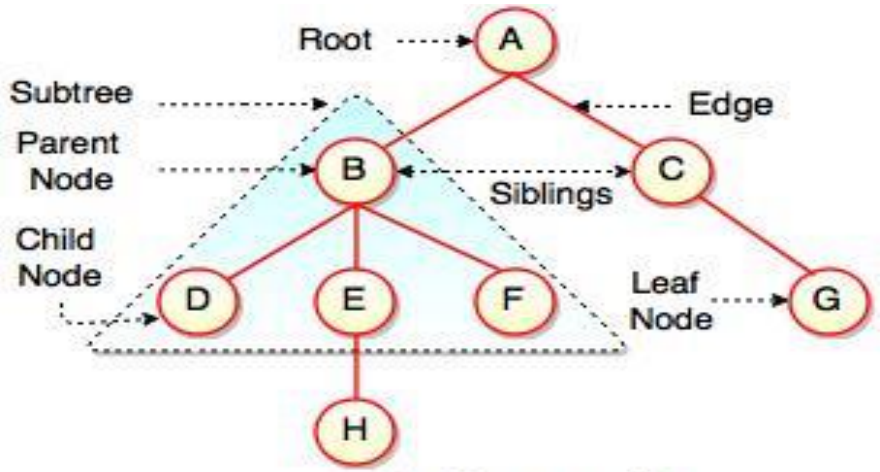


Fig. Structure of Tree

- The above figure represents structure of a tree. Tree has 2 subtrees.
- A is a parent of B and C.
- B is called a child of A and also parent of D, E, F.
- Tree is a collection of elements called Nodes, where each node can have arbitrary number of children.

Basic Terminologies of Trees

Field	Description
Root	Root is a special node in a tree. The entire tree is referenced through it. It does not have a parent.
Parent Node	Parent node is an immediate predecessor of a node.
Child Node	All immediate successors of a node are its children.
Siblings	Nodes with the same parent are called Siblings.
Path	Path is a number of successive edges from source node to destination node.
Height of Node	Height of a node represents the number of edges on the longest path between that node and a leaf.
Height of Tree	Height of tree represents the height of its root node.
Depth of Node	Depth of a node represents the number of edges from the tree's root node to the node.
Degree of Node	Degree of a node represents a number of children of a node.
Edge	Edge is a connection between one node to another. It is a line between two nodes or a node and a leaf.

Node

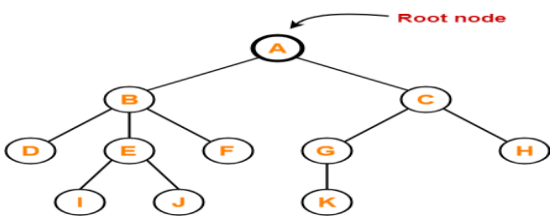
- Node – user-defined data structure that that contains pointers to data and pointers to other nodes
- The code to write a tree node has a data part and references to its left and right child nodes.

```
struct node
 {
   int data;
   struct node *leftChild;
   struct node *rightChild;
};
```

Root-The first node from where the tree originates is called as a **root node**.

- In any tree, there must be only one root node.
- We can never have multiple root nodes in a tree data structure.

Example

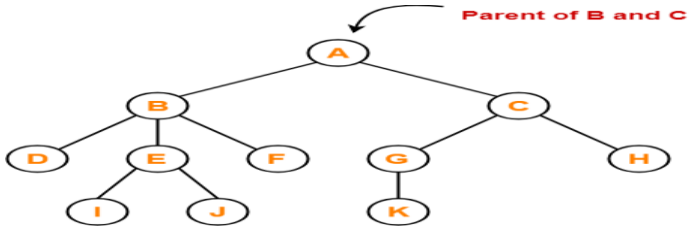


Here, node A is the only root node.

Parent-The node which has a branch from it to any other node is called as a **parent node**.

- In other words, the node which has one or more children is called as a parent node.
- In a tree, a parent node can have any number of child nodes.

Example

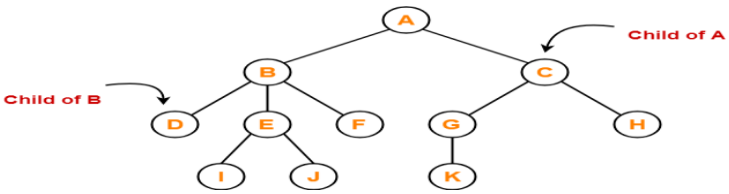


Here,
Node A is the parent of nodes B and C
Node B is the parent of nodes D, E and F

Child-The node which is a descendant of some node is called as a **child node**.

- All the nodes except root node are child nodes.

Example

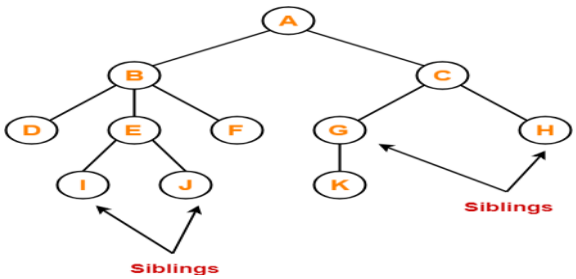


Here,Nodes B and C are the children of node A

Siblings-Nodes which belong to the same parent are called as **siblings**.

- In other words, nodes with the same parent are sibling nodes.

Example

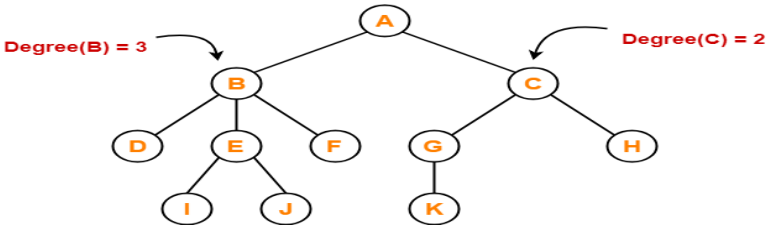


Here, Nodes B and C are siblings

Degree-

- **Degree of a node** is the total number of children of that node.
- **Degree of a tree** is the highest degree of a node among all the nodes in the tree.

Example

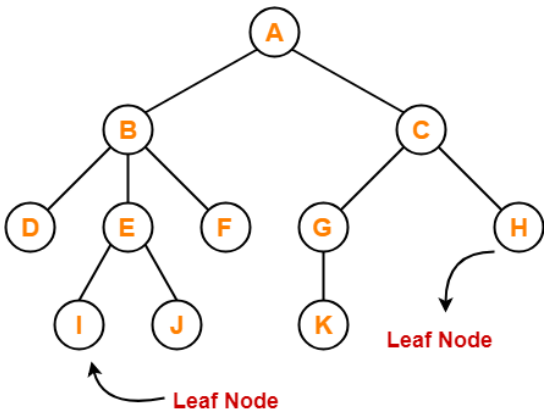


Here,
Degree of node A = 2
Degree of node B = 3
Degree of node C = 2
Degree of node D = 0

Leaf Node-The node which does not have any child is called as a **leaf node**.

- Leaf nodes are also called as **external nodes** or **terminal nodes**.

Example

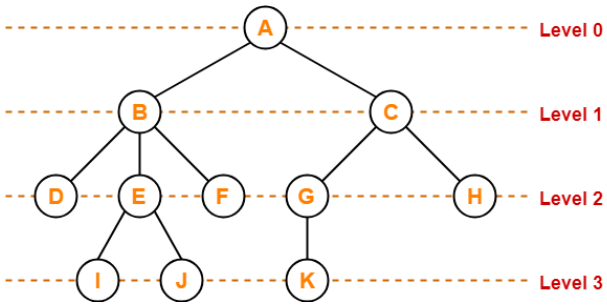


Here, nodes D, I, J, F, K and H are leaf nodes.

Level-In a tree, each step from top to bottom is called as **level of a tree**.

- The level count starts with 0 and increments by 1 at each level or step.

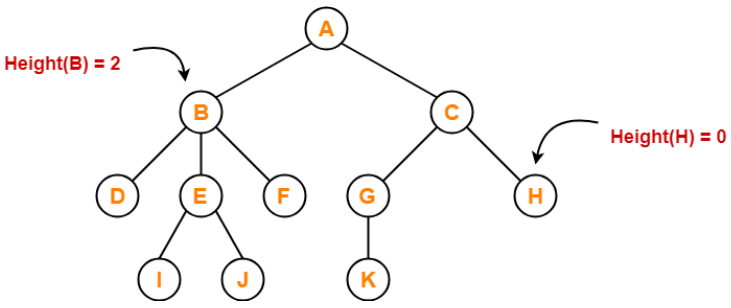
Example



Height-Total number of edges that lies on the longest path from any leaf node to a particular node is called as **height of that node**.

- **Height of a tree** is the height of root node.
- Height of all leaf nodes = 0

Example

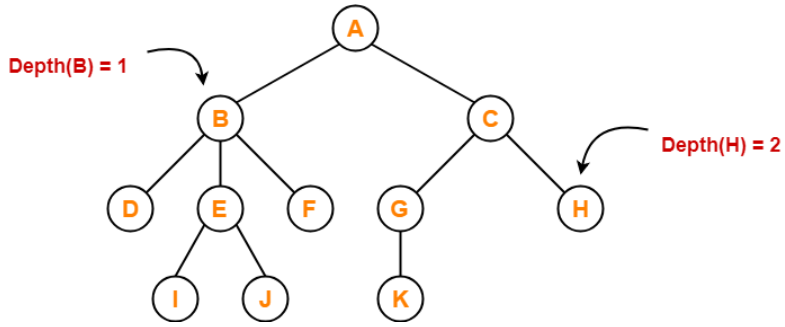


Here,
Height of node A = 3
Height of node B = 2
Height of node C = 2

Depth- Total number of edges from root node to a particular node is called as **depth of that node**.

- **Depth of a tree** is the total number of edges from root node to a leaf node in the longest path.
- Depth of the root node = 0
- The terms “level” and “depth” are used interchangeably.

Example

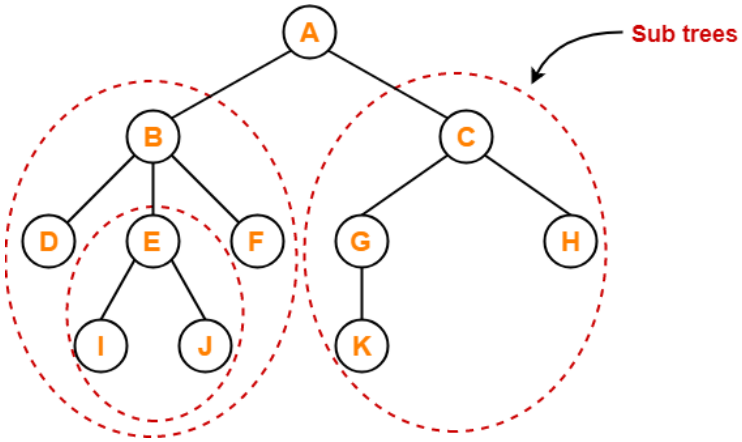


Here,
Depth of node A = 0
Depth of node B = 1
Depth of node C = 1
Depth of node D = 2

Subtree-

In a tree, each child from a node forms a **subtree** recursively.
Every child node forms a subtree on its parent node.

Example



Video Content / Details of website for further learning (if any):
https://www.gatevidyalay.com/tree-data-structure-tree-terminology/
https://www.tutorialride.com/data-structures/trees-in-data-structure.html

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **Page Nos:** 105-107

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-20

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III[TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Binary tree –Representation of binary Tree

Introduction :

- Binary Tree is a special data structure used for data storage purposes.
- A binary tree has a special condition that each node can have a maximum of two children.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Nodes
- Tree
- Non linear structure

Detailed content of the Lecture:

- A binary tree has a root node. It may not have any child nodes (0 child nodes, NULL tree).
- A root node may have one or two child nodes.
- Each node forms a binary tree itself.
- The number of child nodes cannot be more than two.
- It has a unique path from the root to every other node Prefix Expression

1. Full Binary Tree
2. Complete Binary Tree

1. Full Binary Tree

- If each node of binary tree has either two children or no child at all, is said to be a **Full Binary Tree**.
- Full binary tree is also called as **Strictly Binary Tree**.

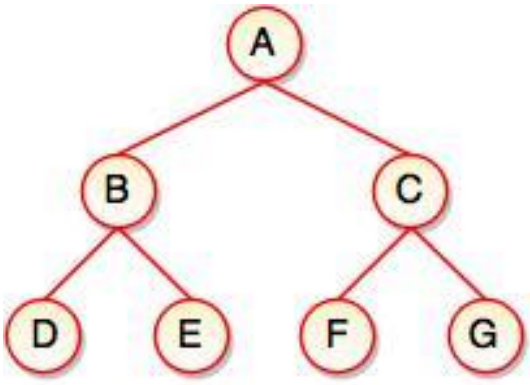


Fig. Full Binary Tree

- Every node in the tree has either 0 or 2 children.
- Full binary tree is used to represent mathematical expressions.

2. Complete Binary Tree

- If all levels of tree are completely filled except the last level and the last level has all keys as left as possible, is said to be a **Complete Binary Tree**.
- Complete binary tree is also called as **Perfect Binary Tree**.

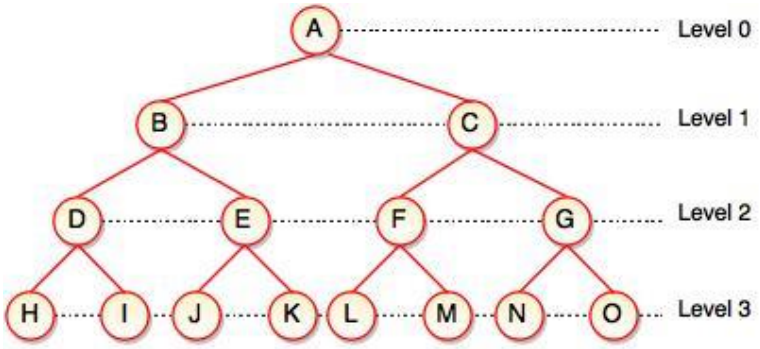


Fig. Complete Binary Tree

In a complete binary tree, every internal node has exactly two children and all leaf nodes are at same level.

For example, at Level 2, there must be $2^2 = 4$ nodes and at Level 3 there must be $2^3 = 8$ nodes.

Properties of Binary Trees:

Some of the important properties of a binary tree are as follows:

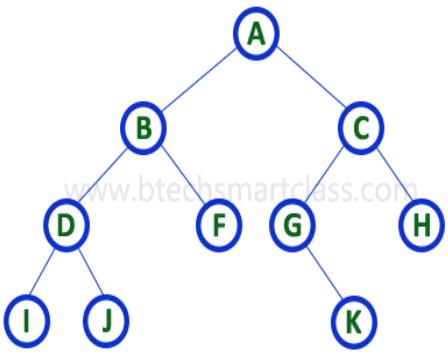
1. If *h* = height of a binary tree, then
2. Maximum number of leaves = 2^h
3. Maximum number of nodes = $2^{h+1} - 1$
4. If a binary tree contains m nodes at level l, it contains at most 2m nodes at level l + 1.
5. Since a binary tree can contain at most one node at level 0 (the root), it can contain at most 2^l node at level l.
6. The total number of edges in a full binary tree with n node is n - 1.

A binary tree data structure is represented using two methods. Those methods are as follows...

Array Representation

Linked List Representation

Consider the following binary tree...



1.Array Representation of Binary Tree

- In array representation of a binary tree, one-dimensional array (1-D Array) to represent a binary tree.
- Consider the above example of a binary tree and it is represented as follows...

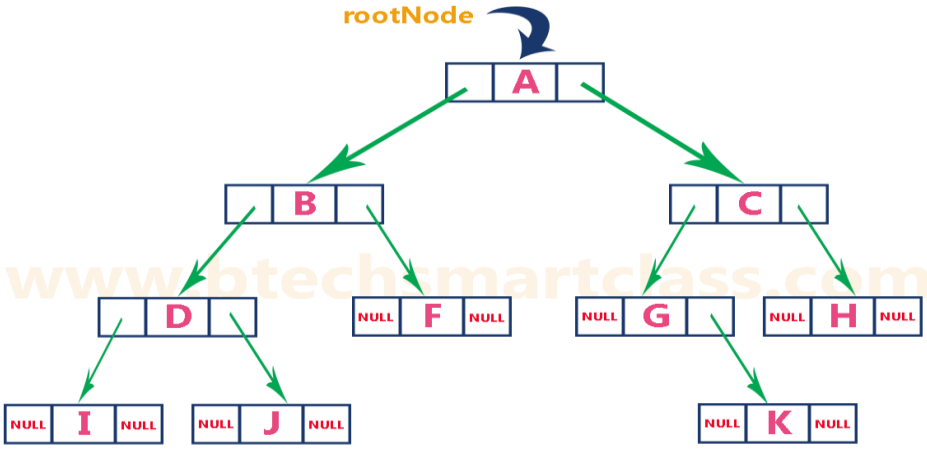
A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

To represent a binary tree of depth '**n**' using array representation, we need one dimensional array with a maximum size of **2n + 1**.

2.Linked List Representation of Binary Tree

- It uses a double linked list to represent a binary tree.
- In a double linked list, every node consists of three fields.
- First field for storing left child address, second for storing actual data and third for storing right child address.
- In this linked list representation, a node has the following structure...

The above example of the binary tree represented using Linked list representation is shown as follows.



Video Content / Details of website for further learning (if any):
https://www.youtube.com/watch?v=sFVxsglODoo
https://nptel.ac.in/courses/106/106/106106133/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **Page Nos:** 105-107

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-21

LECTURE HANDOUTS

IT

II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III[TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Binary Tree Traversal

Introduction :

- Traversal of a binary tree means to visit each node in the tree exactly once. The tree traversal is used in allt it.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Data structure
- Node
- Tree
- Subtree

Detailed content of the Lecture:

- Traversal is a process to visit all the nodes of a tree and may print their values too.
- Because, all nodes are connected via edges (links) we always start from the root (head) node.
- It cannot be randomly access a node in a tree.

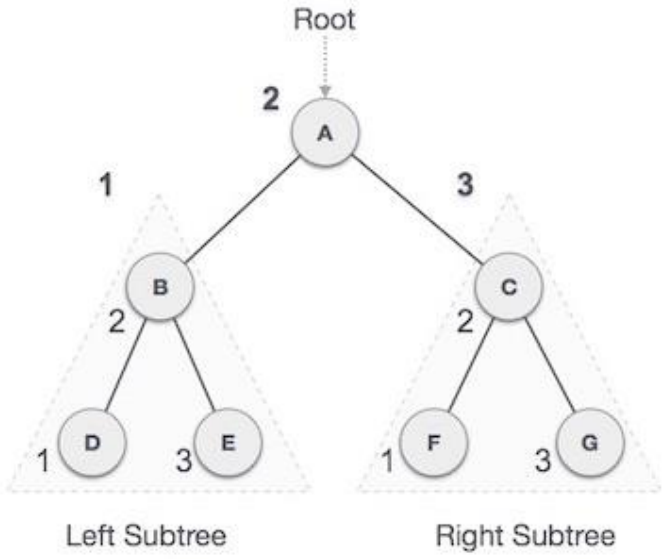
There are three ways which we use to traverse a tree

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

In-order Traversal

- In this traversal method, the left subtree is visited first, then the root and later the right sub-tree.
- It should always remember that every node may represent a subtree itself.
- If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.



- Start from **A**, and following in-order traversal, we move to its left subtree **B**.
- **B** is also traversed in-order.
- The process goes on until all the nodes are visited.

The output of inorder traversal of this tree will be −

D → B → E → A → F → C → G

Algorithm

Until all nodes are traversed

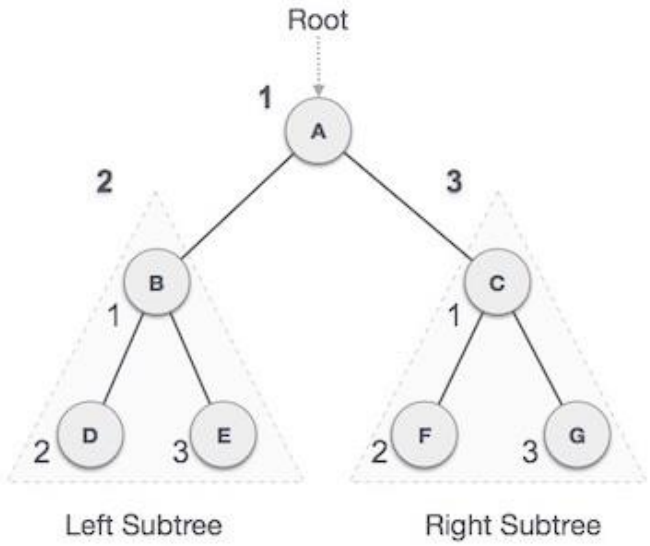
Step 1 − Recursively traverse left subtree.

Step 2 − Visit root node.

Step 3 − Recursively traverse right subtree.

Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



- Start from **A**, and following pre-order traversal, first visit **A** itself and then move to its left subtree **B**.
- **B** is also traversed pre-order.
- The process goes on until all the nodes are visited.

The output of pre-order traversal of this tree will be

A → B → D → E → C → F → G

Algorithm

Until all nodes are traversed

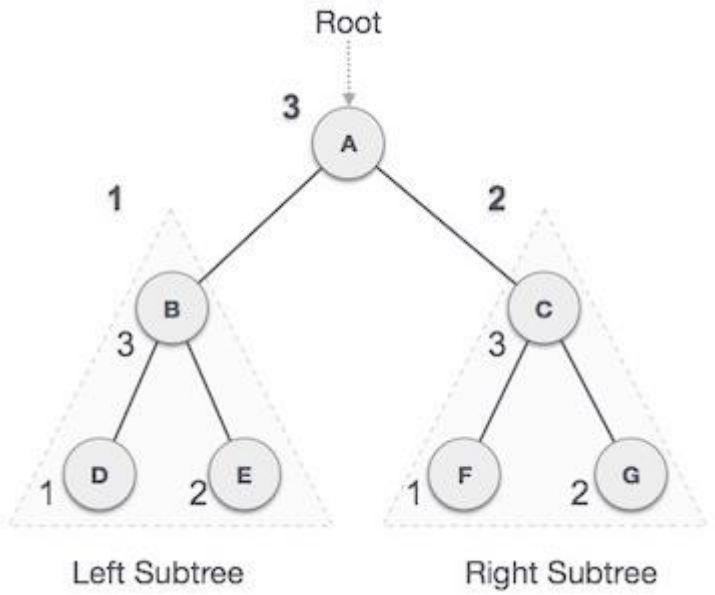
Step 1 − Visit root node.

Step 2 − Recursively traverse left subtree.

Step 3 − Recursively traverse right subtree.

Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First traverse the left subtree, then the right subtree and finally the root node.



- Start from **A**, and following Post-order traversal, first visit the left subtree **B**.
- **B** is also traversed post-order.
- The process goes on until all the nodes are visited.

The output of post-order traversal of this tree will be

$$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$$

Algorithm

Until all nodes are traversed −

Step 1 − Recursively traverse left subtree.

Step 2 − Recursively traverse right subtree.

Step 3 − Visit root node.

Video Content / Details of website for further learning (if any):

https://www.tutorialspoint.com/data_structures_algorithms/tree_traversal.htm

Important Books/Journals for further learning including the page nos.:

E.Horowitz,S.Sahni Susan , Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008-**Page Nos:** 19-22

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-22**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III[TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Operations on Binary tree, Tree representation of an arithmetic expression

Introduction :

- Traversal of a binary tree means to visit each node in the tree exactly once. The tree traversal is used in allt it.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Data structure
- Node
- Tree
- Subtree

Detailed content of the Lecture:

Operations on a Binary Search Tree

The following operations are performed on a binary search tree.

- **Search**
- **Insertion**
- **Deletion**

Search Operation in BST

Step 1 - Read the search element from the user.

Step 2 - Compare the search element with the value of root node in the tree.

Step 3 - If both are matched, then display "Given node is found!!!" and terminate the function

Step 4 - If both are not matched, then check whether search element is smaller or larger than that node value.

Step 5 - If search element is smaller, then continue the search process in left subtree.

Step 6- If search element is larger, then continue the search process in right subtree.

Step 7 - Repeat the same until we find the exact element or until the search element is compared with the leaf node

Insertion Operation in BST

- In a binary search tree, the insertion operation is performed with **O(log n)** time complexity.
- In binary search tree, new node is always inserted as a leaf node.

The insertion operation is performed as follows.

Step 1 - Create a newNode with given value and set its **left** and **right** to **NULL**.

Step 2 - Check whether tree is Empty.

Step 3 - If the tree is **Empty**, then set **root** to **newNode**.

Step 4 - If the tree is **Not Empty**, then check whether the value of newNode is **smaller** or **larger** than the node (here it is root node).

Step 5 - If newNode is **smaller** than **or equal** to the node then move to its **left** child.

Step 6- Repeat the above steps until we reach to the **leaf** node (i.e., reaches to NULL).

Step 7 - After reaching the leaf node, insert the newNode as **left child** if the newNode is **smaller or equal** to that leaf node or else insert it as **right child**.

Deletion Operation in BST

- In a binary search tree, the deletion operation is performed with **O(log n)** time complexity.

Deleting a node from Binary search tree includes following three cases.

Case 1: Deleting a Leaf node (A node with no children)

Case 2: Deleting a node with one child

Case 3: Deleting a node with two children

Case 1: Deleting a leaf node

The following steps to delete a leaf node from BST...

Step 1 - Find the node to be deleted using **search operation**

Step 2 - Delete the node using **free** function (If it is a leaf) and terminate the function.

Case 2: Deleting a node with one child

The following steps to delete a node with one child from BST...

Step 1 - Find the node to be deleted using **search operation**

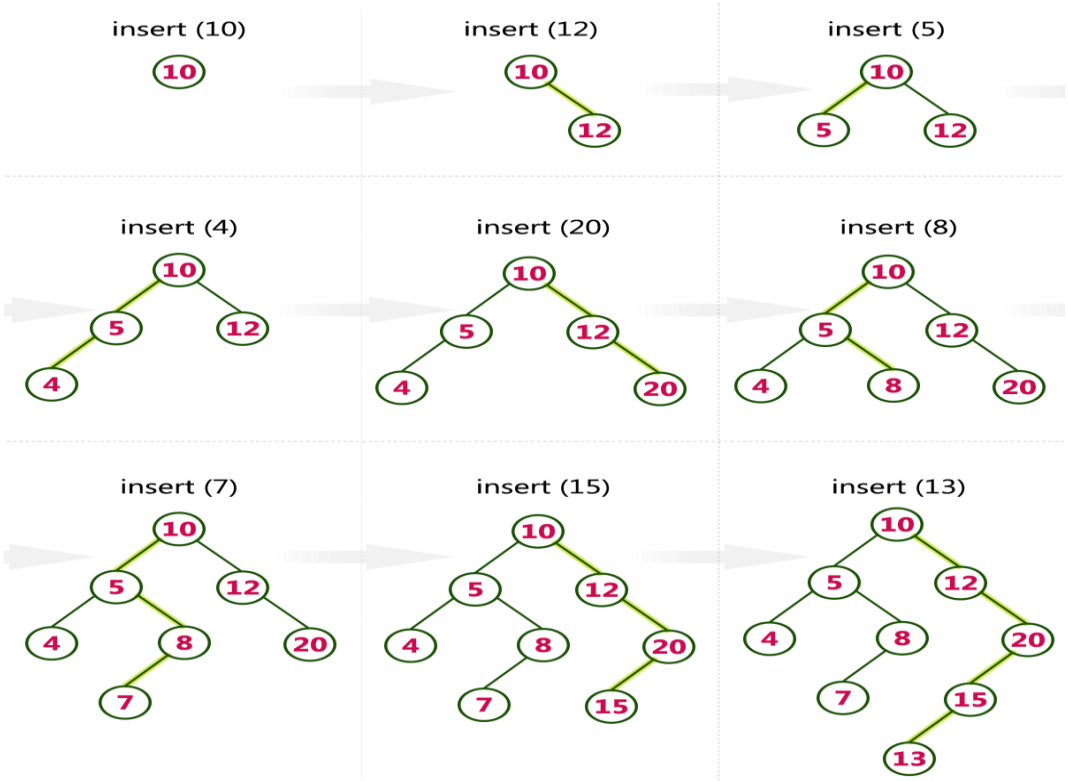
Step 2 - If it has only one child then create a link between its parent node and child node.

Step 3 - Delete the node using **free** function and terminate the function.

Construct a Binary Search Tree by inserting the following sequence of numbers...

10,12,5,4,20,8,7,15 and 13

Above elements are inserted into a Binary Search Tree as follows. Tree representation of an arithmetic



expression

- An **expression** is a string of symbols. **Arithmetic expressions** are made up of variable names, binary operators and brackets
- Example Arithmetic Expression: A + (B * (C / D))
- Tree for the above expression:
- Leaves = operands (constants/variables)
- Non-leaf nodes = operators

  + A * B / C D

Video Content / Details of website for further learning (if any):
http://www.btechsmartclass.com/data_structures/binary-search-tree.html

Important Books/Journals for further learning including the page nos.:
E.Horowitz,S.Sahni Susan , Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008-**page nos:**23-25

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-23**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III[TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Binary Search Tree – Creation, Searching for an item

Introduction :

- The nodes of a binary tree can be numbered in a natural way, level by level, left to right.
- The nodes of a complete binary tree can be numbered so that the root is assigned the number 1, a left child is assigned twice the number assigned its parent, and a right child is assigned one more than twice the number assigned its parent

Prerequisite knowledge for Complete understanding and learning of Topic:

- Node
- Path
- Root

Detailed content of the Lecture:

Binary Search Tree is a special kind of binary tree in which nodes are arranged in a specific order.

In a binary search tree (BST), each node contains-

Only smaller values in its left sub tree

Only larger values in its right sub tree

Binary Search Tree Construction-

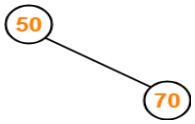
Construct a Binary Search Tree (BST) for the following sequence of numbers-

50, 70, 60, 20, 90, 10, 40

Insert **50**

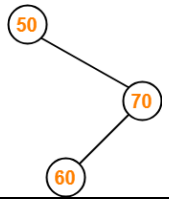
Insert **70**

As 70 > 50, so insert 70 to the right of 50.



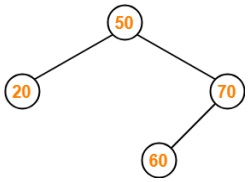
Insert **60**

As 60 > 50, so insert 60 to the right of 50. As 60 < 70, so insert 60 to the left of 70.



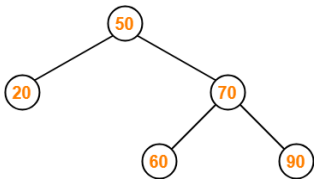
Insert **20**

As 20 < 50, so insert 20 to the left of 50.



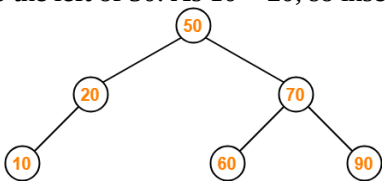
Insert **90**

As 90 > 50, so insert 90 to the right of 50. As 90 > 70, so insert 90 to the right of 70.



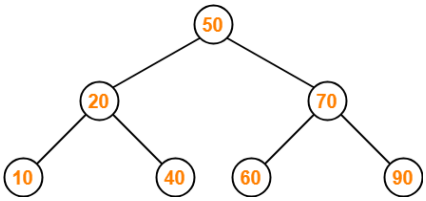
Insert **10**

As 10 < 50, so insert 10 to the left of 50. As 10 < 20, so insert 10 to the left of 20.



Insert **40**

As 40 < 50, so insert 40 to the left of 50. As 40 > 20, so insert 40 to the right of 20.



Video Content / Details of website for further learning (if any):
https://www.gatevidyalay.com/binary-search-trees-data-structures/
https://www.youtube.com/watch?v=Qat40osl21g

Important Books/Journals for further learning including the page nos.:
R.Kruse,C.L.Tondo and B.Leung,Data structures and Program Design in C, 2nd Edition , Prentice-Hall, 2006,**page nos:** 116,123

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L-24**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III[TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Minimum, Maximum or any value

Introduction :

- The left sub-tree of a node contains only nodes with keys less than the node's key.
- The right sub-tree of a node contains only nodes with keys greater than the node's key.
- The left and right sub-tree each must also be a binary search tree.
  There must be no duplicate nodes.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Two important topics)

- Searching
- Minimum VALUES
- Maximum Values
- Applications of Binary Search tree

Detailed content of the Lecture:

- A binary search tree is a special binary tree, which is either empty or it should satisfy the following characteristics: Every node has a value and no two nodes should have the same value i.e) the values in the binary search tree are distinct ·
- The values in any left sub-tree is less than the value of its parent node ·
- The values in any right sub-tree is greater than the value of its parent node ·
- The left and right sub-trees of each node are again binary

- The left sub-tree of a node contains only nodes with keys less than the node's key.
- The right sub-tree of a node contains only nodes with keys greater than the node's key.

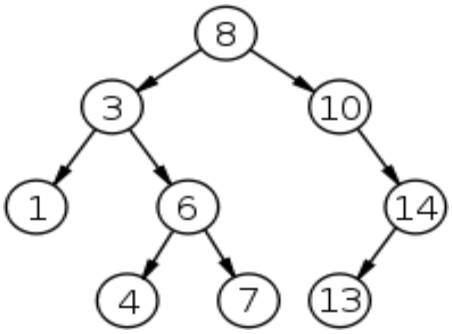
The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

the values in the binary search tree are distinct

· The values in any left sub-tree is less than the value of its parent node

· The values in any right sub-tree is greater than the value of its parent node

· The left and right sub-trees of each node are again binary search trees



Searching a key

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right sub-tree of root node. Otherwise we recur for left sub-tree.

A utility function to search a given key in
BST def search(root,key):

```
  # Base Cases: root is null or key is present at root
  if root is None or root.val ==
     key: return root
   # Key is greater than root's
  key if root.val < key:
     return search(root.right,key)
    # Key is smaller than root's
 key return
 search(root.left,key)
```

APPLICATIONS OF BINARY SEARC TREE

- Calls to large companies
- Access to limited resources in Universities
- Accessing files from file server

Video Content / Details of website for further learning (if any):
http://www.btechsmartclass.com/data_structures/binary-search-tree.html
http://www.btechsmartclass.com/binary-search-tree.html

Important Books/Journals for further learning including the page nos.:
R.Kruse,C.L.Tondo and B.Leung,Data structures and Program Design in C, 2nd Edition , Prentice-Hall, 2006,**Page Nos:** 116-123

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-25**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III[TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Applications of Binary search tree, MaxHeap-Definition

Introduction :

- The left sub-tree of a node contains only nodes with keys less than the node's key.
- The right sub-tree of a node contains only nodes with keys greater than the node's key.
- The left and right sub-tree each must also be a binary search tree.
  There must be no duplicate nodes.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Two important topics)

- Searching
- Minimum VALUES
- Maximum Values
- Applications of Binary Search tree

Detailed content of the Lecture:

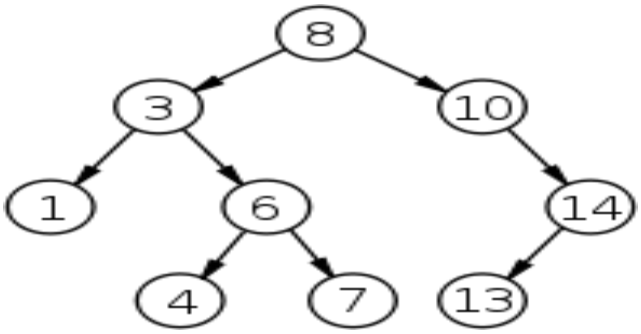
- A binary search tree is a special binary tree, which is either empty or it should satisfy the following characteristics: Every node has a value and no two nodes should have the same value i.e) the values in the binary search tree are distinct ·
- The values in any left sub-tree is less than the value of its parent node ·
- The values in any right sub-tree is greater than the value of its parent node ·
- The left and right sub-trees of each node are again binary
- The left sub-tree of a node contains only nodes with keys less than the node's key.
- The right sub-tree of a node contains only nodes with keys greater than the node's key.

The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.the values in the binary search tree are distinct

· The values in any left sub-tree is less than the value of its parent node

· The values in any right sub-tree is greater than the value of its parent node

· The left and right sub-trees of each node are again binary search trees



Searching a key

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right sub-tree of root node. Otherwise we recur for left sub-tree.

```
# A utility function to search a given key in
BST def search(root,key):

    # Base Cases: root is null or key is present at root
    if root is None or root.val ==
        key: return root
     # Key is greater than root's
    key if root.val < key:
        return search(root.right,key)
       # Key is smaller than root's
   key return
   search(root.left,key)
```

APPLICATIONS OF BINARY SEARC TREE

- Calls to large companies
- Access to limited resources in Universities
- Accessing files from file server

Video Content / Details of website for further learning (if any):
http://www.btechsmartclass.com/data_structures/binary-search-tree.html
http://www.btechsmartclass.com/binary-search-tree.html

Important Books/Journals for further learning including the page nos.:
R.Kruse,C.L.Tondo and B.Leung,Data structures and Program Design in C, 2nd Edition , Prentice-Hall, 2006,**page nos:** 116-123

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-26**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III[TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Insertion into a Max Heap

Introduction :

- Heap data structure is a specialized binary tree-based data structure.
- Heap is a binary tree with special characteristics. In a heap data structure, nodes are arranged based on their values.
- A heap data structure sometimes also called as Binary Heap.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Heap
- Node
- Binary Heap

Detailed content of the Lecture:

Max Heap

Max heap data structure is a specialized full binary tree data structure. In a max heap nodes are arranged based on node value. Max heap is defined as follows

Max heap is a specialized full binary tree in which every parent node contains greater or equal value than its child nodes.

Insertion Operation in Max Heap

Insertion Operation in max heap is performed as follows...

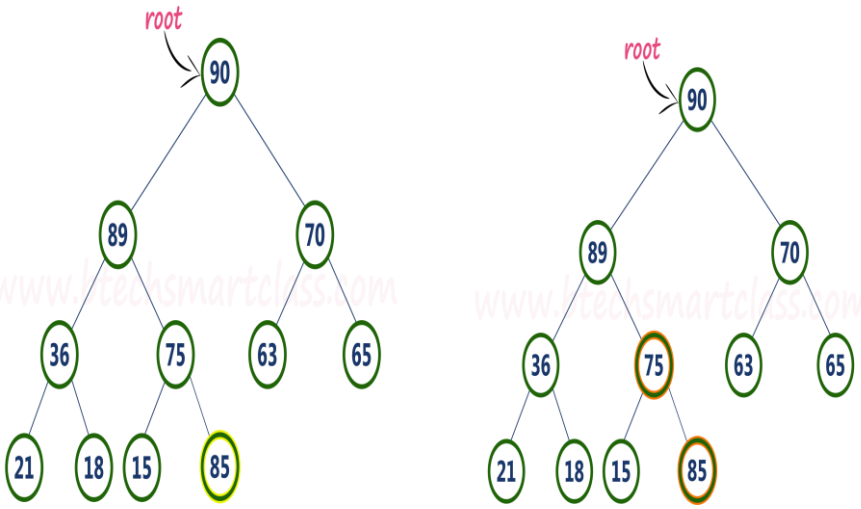
Step 1 - Insert the **newNode** as **last leaf** from left to right.

Step 2 - Compare **newNode value** with its **Parent node**.

Step 3 - If **newNode value is greater** than its parent, then **swap** both of them.

Step 4 - Repeat step 2 and step 3 until newNode value is less than its parent node (or) newNode reaches to root.Consider the above max heap. **Insert a new node with value 85.**

Step 1 - Insert the **newNode** with value 85 as **last leaf** from left to right. That means newNode is added as a right child of node with value 75. After adding max heap is as follows...



Step 2 - Compare **newNode value (85)** with its **Parent node value (75)**. That means **85 > 75**

Step 3 - Here **newNode value (85) is greater** than its **parent value (75)**, then **swap** both of them. After swapping, max heap is as follows.

Deletion Operation in Max Heap

In a max heap, deleting the last node is very simple as it does not disturb max heap properties. Deleting root node from a max heap is little difficult as it disturbs the max heap properties. We use the following steps to delete the root node from a max heap...

Step 1 - **Swap** the **root** node with **last** node in max heap

Step 2 - **Delete** last node.

Step 3 - Now, compare **root value** with its **left child value**.

Step 4 - If **root value is smaller** than its left child, then compare **left child** with its **right sibling**. Else goto **Step 6**

Step 5 - If **left child value is larger** than its **right sibling**, then **swap root** with **left child** otherwise **swap root** with its **right child**.

Step 6 - If **root value is larger** than its left child, then compare **root value** with its **right child** value.

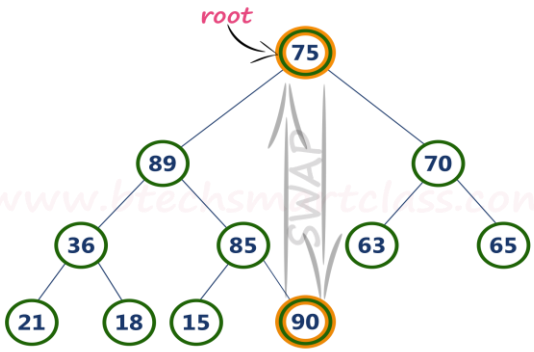
Step 7 - If **root value is smaller** than its **right child**, then **swap root** with **right child** otherwise **stop the process**.

Step 8 - Repeat the same until root node fixes at its exact position.

Example

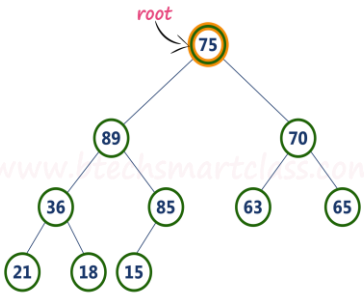
Consider the above max heap. **Delete root node (90) from the max heap.**

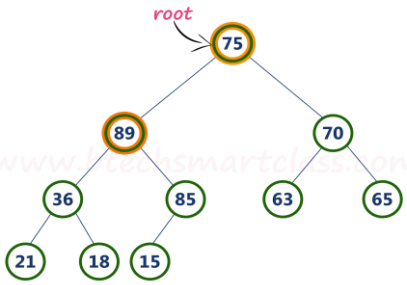
Step 1 - Swap the **root node (90)** with **last node 75** in max heap. After swapping max heap is as follows



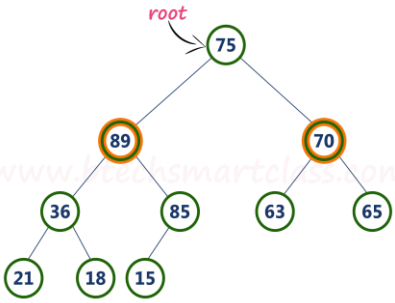
Step 2 - Delete last node. Here the last node is 90. After deleting node with value 90 from heap, max heap is as follows

Step 3 - Compare **root node (75)** with its **left child (89)**.





Here, **root value (75) is smaller** than its left child value (89). So, compare left child (89) with its right sibling (70).



Video Content / Details of website for further learning (if any):
http://www.btechsmartclass.com/data_structures/max-heap.html
http://www.btechsmartclass.com/data_structures/binary-heap.html

Important Books/Journals for further learning including the page nos.:
R.Kruse,C.L.Tondo and B.Leung,Data structures and Program Design in C, 2nd Edition , Prentice-Hall, 2006,page nos: 193-207

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L-27**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : III[TREE AND BINARY SEARCH TREE] Date of Lecture:

Topic of Lecture: Deletion from a Max Heap

Introduction :

- Heap data structure is a specialized binary tree-based data structure.
- Heap is a binary tree with special characteristics. In a heap data structure, nodes are arranged based on their values.
- A heap data structure sometimes also called as Binary Heap.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Heap
- Node
- Binary Heap

Detailed content of the Lecture:

Max Heap

Max heap data structure is a specialized full binary tree data structure. In a max heap nodes are arranged based on node value.Max heap is defined as follows

Max heap is a specialized full binary tree in which every parent node contains greater or equal value than its child nodes.

Insertion Operation in Max Heap

Insertion Operation in max heap is performed as follows...

Step 1 - Insert the **newNode** as **last leaf** from left to right.

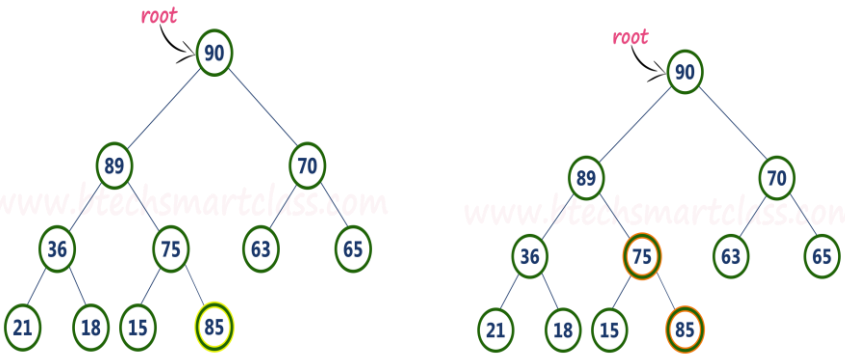
Step 2 - Compare **newNode value** with its **Parent node**.

Step 3 - If **newNode value is greater** than its parent, then **swap** both of them.

Step 4 - Repeat step 2 and step 3 until newNode value is less than its parent node (or) newNode reaches to root.

Consider the above max heap. **Insert a new node with value 85.**

Step 1 - Insert the **newNode** with value 85 as **last leaf** from left to right. That means newNode is added as a right child of node with value 75. After adding max heap is as follows



Step 2 - Compare **newNode value (85)** with its **Parent node value (75)**. That means **85 > 75**

Step 3 - Here **newNode value (85) is greater** than its **parent value (75)**, then **swap** both of them. After swapping, max heap is as follows.

Deletion Operation in Max Heap

In a max heap, deleting the last node is very simple as it does not disturb max heap properties. Deleting root node from a max heap is little difficult as it disturbs the max heap properties. We use the following steps to delete the root node from a max heap...

Step 1 - Swap the **root** node with **last** node in max heap

Step 2 - Delete last node.

Step 3 - Now, compare **root value** with its **left child value**.

Step 4 - If **root value is smaller** than its left child, then compare **left child** with its **right sibling**. Else goto **Step 6**

Step 5 - If **left child value is larger** than its **right sibling**, then **swap root** with **left child** otherwise **swap root** with its **right child**.

Step 6 - If **root value is larger** than its left child, then compare **root value** with its **right child** value.

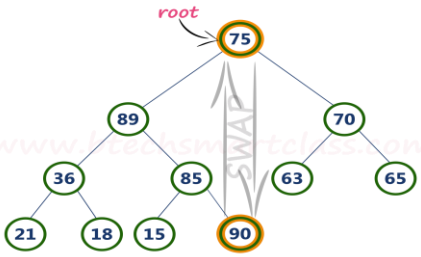
Step 7 - If **root value is smaller** than its **right child**, then **swap root** with **right child** otherwise **stop the process**.

Step 8 - Repeat the same until root node fixes at its exact position.

Example

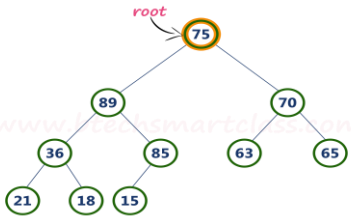
Consider the above max heap. **Delete root node (90) from the max heap.**

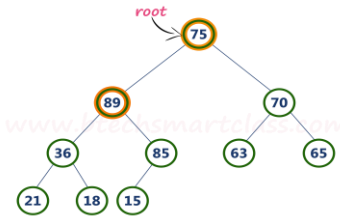
- **Step 1 - Swap** the **root node (90)** with **last node 75** in max heap. After swapping max heap is as follows...



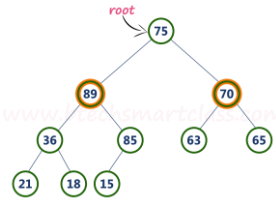
Step 2 - Delete last node. Here the last node is 90. After deleting node with value 90 from heap, max heap is as follows...

Step 3 - Compare **root node (75)** with its **left child (89)**.





Here, **root value (75) is smaller** than its left child value (89). So, compare left child (89) with its right sibling (70).



Video Content / Details of website for further learning (if any):

http://www.btechsmartclass.com/data_structures/max-heap.html

http://www.btechsmartclass.com/data_structures/binary-heap.html

Important Books/Journals for further learning including the page nos.:

R.Kruse,C.L.Tondo and B.Leung,Data structures and Program Design in C, 2nd Edition , Prentice-Hall, 2006,page nos: 193-207

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-28**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Definition – Graph terminologies

Introduction :

- Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs). Here edges are used to connect the vertices.

Prerequisite knowledge for Complete understanding and learning of Topic:

- non-linear data structure
- nodes

Detailed content of the Lecture:

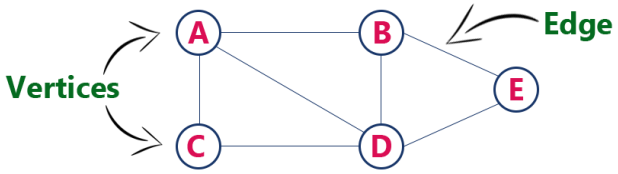
- Graph is a collection of vertices and arcs in which vertices are connected with arcs
- Graph is a collection of nodes and edges in which nodes are connected with edges
- A graph **G** is represented as **G = (V , E)**, where **V is set of vertices** and **E is set of edges**.

Example

The following is a graph with 5 vertices and 6 edges.

This graph G can be defined as G = (V , E)

Where V = {A,B,C,D,E} and E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.



Graph Terminology

Vertex

- Individual data element of a graph is called as Vertex. **Vertex** is also known as **node**.
- In above example graph, A, B, C, D & E are known as vertices.

Edge

- An edge is a connecting link between two vertices. **Edge** is also known as **Arc**. An edge is represented as (startingVertex, endingVertex).
- For example, in above graph the link between vertices A and B is represented as (A,B).
- In above example graph, there are 7 edges (i.e., (A,B), (A,C), (A,D), (B,D), (B,E), (C,D), (D,E)).

Edges are three types.

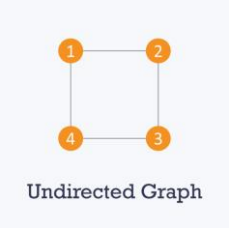
Undirected Edge - An undirected egde is a bidirectional edge. If there is undirected edge between vertices A and B then edge (A , B) is equal to edge (B , A).

Directed Edge - A directed egde is a unidirectional edge. If there is directed edge between vertices A and B then edge (A , B) is not equal to edge (B , A).

Weighted Edge - A weighted egde is a edge with value (cost) on it.

Undirected graph

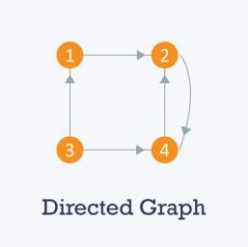
Undirected: An undirected graph is a graph in which all the edges are bi-directional i.e. the edges do not point in any specific direction.



Undirected Graph

Directed graph

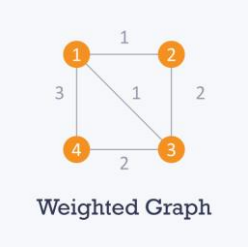
Directed: A directed graph is a graph in which all the edges are uni-directional i.e. the edges point in a single direction.



Directed Graph

Weighted Graph

A graph (or digraph) is termed as weighted graph if all edges in it are labeled with some weights. Eg:



Weighted Graph

Video Content / Details of website for further learning (if any):

http://www.btechsmartclass.com/data_structures/introduction-to-graphs.html

https://nptel.ac.in/courses/106106133/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 299-300

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L-29**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Graph Terminologies

Introduction :

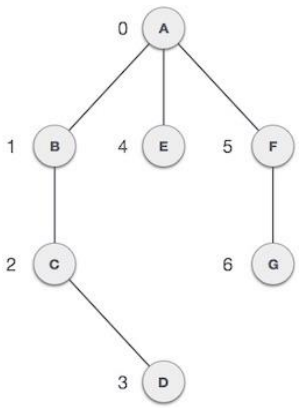
- Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs). Here edges are used to connect the vertices.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Non-Linear Data Structure
- Nodes
- Edges
- Vertices

Detailed content of the Lecture:

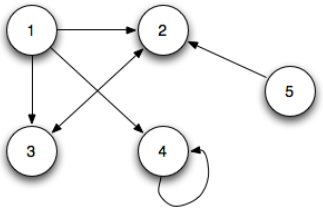
Adjacent Vertices

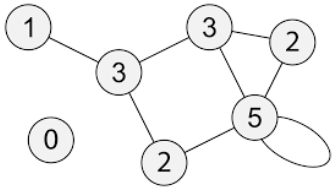


Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.

Self-loop

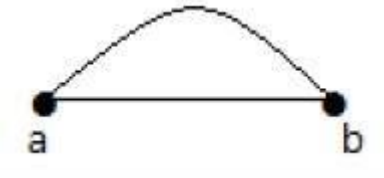
Edge (undirected or directed) is a self-loop if its two endpoints coincide with each other.





Parallel edges or Multiple edges

If there are two undirected edges with same end vertices and two directed edges with same origin and destination, such edges are called parallel edges or multiple edges.



Path

A path is a sequence of alternate vertices and edges that starts at a vertex and ends at other vertex such that each edge is incident to its predecessor and successor vertex.

Cycle

Cycle is a path that starts and end at the same vertex.

Degree

Total number of edges connected to a vertex is said to be degree of that vertex.

Indegree

Total number of incoming edges connected to a vertex is said to be indegree of that vertex.

Outdegree

Total number of outgoing edges connected to a vertex is said to be outdegree of that vertex.

Connected Graph

Two vertices vi, vj in a graph G is said to be connected only if there is a path in G between vi and vj.

Complete Graph

An n vertex undirected graph with exactly n(n-1)/2 edges is said to be complete graph. The graph G is said to be complete graph .

Video Content / Details of website for further learning (if any):

http://www.btechsmartclass.com/data_structures/introduction-to-graphs.html

https://nptel.ac.in/courses/106106133/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 299-300

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-30**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Representation of Graph

Introduction :

- To represent a graph, we just need the set of vertices, and for each vertex the neighbors of the vertex (vertices which is directly connected to it by an edge).
- If it is a weighted graph, then the weight will be associated with each edge.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Graph
- Edges
- Vertices

Detailed content of the Lecture:

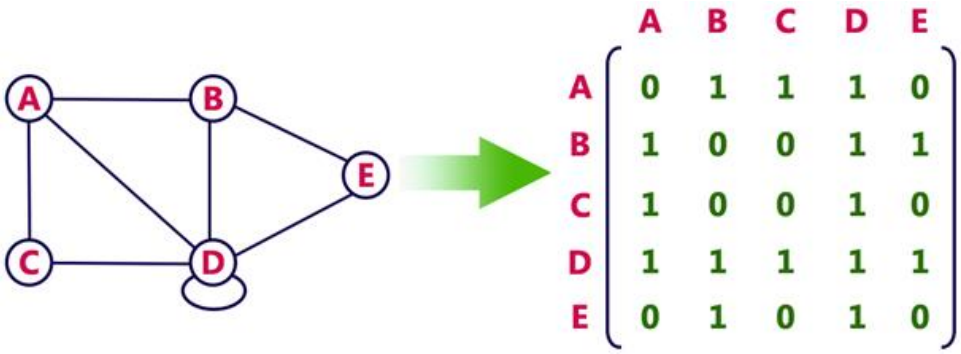
Adjacency Matrix

- Adjacency matrix is a sequential representation.
- It is used to represent which nodes are adjacent to each other. i.e. is there any edge connecting nodes to a graph.
- In this representation, we have to construct a nXn matrix A.
- If there is any edge from a vertex i to vertex j, then the corresponding element of A, $a^{i,j} = 1$, otherwise $a^{i,j} = 0$.
- If there is any weighted graph then instead of 1s and 0s, it can store the weight of the edge.

Example

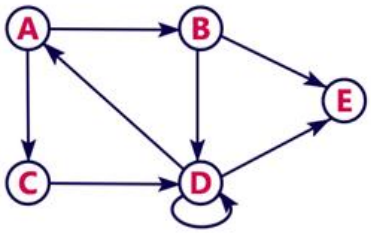
Consider the following **undirected graph representation**:

Undirected graph representation



Directed graph representation

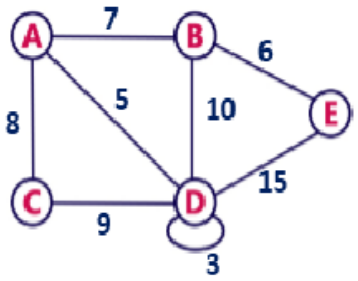
See the directed graph representation:



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

In the above examples, 1 represents an edge from row vertex to column vertex, and 0 represents no edge from row vertex to column vertex.

Undirected weighted graph representation



	A	B	C	D	E
A	0	7	8	5	0
B	7	0	0	10	6
C	8	0	0	9	0
D	5	10	9	3	15
E	0	6	0	15	0

Pros: Representation is easier to implement and follow.

Cons: It takes a lot of space and time to visit all the neighbors of a vertex, we have to traverse all the vertices in the graph, which takes quite some time.

Video Content / Details of Itbsite for further learning (if any):

http://www.btechsmartclass.com/data_structures/graph-representations.html

https://nptel.ac.in/courses/106/106/106106133/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 300-301

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L-31**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Representation of Graph Linked representation

Introduction :

- To represent a graph, we just need the set of vertices, and for each vertex the neighbors of the vertex (vertices which is directly connected to it by an edge).
- If it is a weighted graph, then the weight will be associated with each edge.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Graph
- Edges
- Vertices

Detailed content of the Lecture:

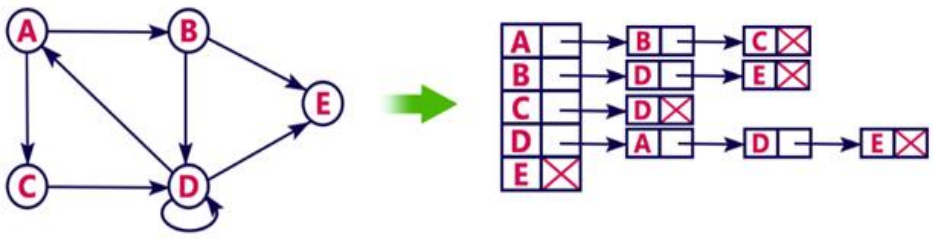
Adjacency List

Adjacency list is a linked representation.

- In this representation, for each vertex in the graph, we maintain the list of its neighbors. It means, every vertex of the graph contains list of its adjacent vertices.
- An array of vertices which is indexed by the vertex number and for each vertex v, the corresponding array element points to a **singly linked list** of neighbors of v.

Example

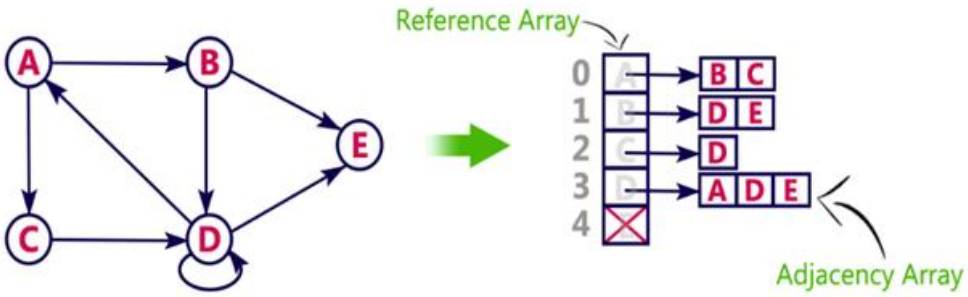
Let's see the following directed graph representation implemented using linked list:



Adjacency list is a Array representation

- An array of lists is used. Size of the array is equal to the number of vertices.
- Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the ith vertex.
- This representation can also be used to represent a weighted graph.
- The weights of edges can be represented as lists of pairs.

- This representation can also be implemented using an array as follows.




```
struct node
{
   int vertex;
   struct node* next;
};

struct Graph
{
   int numVertices;
   struct node** adjLists;
};
```

Pros:

- Adjacency list saves lot of space.
- We can easily insert or delete as we use linked list.
- Such kind of representation is easy to follow and clearly shows the adjacent nodes of node.

Cons:

- The adjacency list allows testing whether two vertices are adjacent to each other but it is slower to support this operation.

Video Content / Details of Itbsite for further learning (if any):

http://www.btechsmartclass.com/data_structures/graph-representations.html

https://nptel.ac.in/courses/106/106/106106133/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 300-301

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L-32**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Comparison of representations

Introduction :

- To represent a graph, we just need the set of vertices, and for each vertex the neighbors of the vertex (vertices which is directly connected to it by an edge).
- If it is a weighted graph, then the weight will be associated with each edge.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Two important topics)

- Graph
- Linked list Representation
- Array Representation

Detailed content of the Lecture:

Adjacency list

- Vertices are stored as records or objects, and every vertex stores a list of adjacent vertices.This data structure allows the storage of additional data on the vertices.
- Additional data can be stored if edges are also stored as objects, in which case each vertex stores its incident edges and each edge stores its incident vertices.

Adjacency matrix

- A two-dimensional matrix, in which the rows represent source vertices and columns represent destination vertices. Data on edges and vertices must be stored externally.
- Only the cost for one edge can be stored between each pair of vertices.

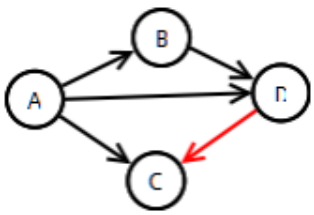
Incidence matrix

- A two-dimensional Boolean matrix, in which the rows represent the vertices and columns represent the edges. The entries indicate whether the vertex at a row is incident to the edge at a column.

Directed graph

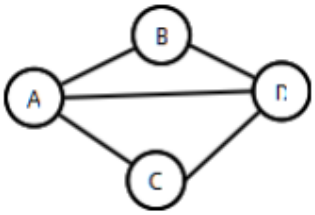
Digraph:

A graph whose edges are directed (i.e have a direction)
Edge drawn as arrow
Edge can only be traversed in direction of arrow
Example: E = {(A,B), (A,C), (A,D), (B,C), (D,C)}



Undirected Graph

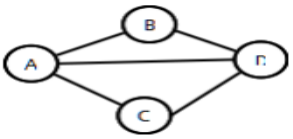
A graph where there is no implied direction on edge between nodes



- In diagrams, edges have no direction (i.e they are not arrows) Can traverse edges in either directions
- Adjacency matrix representation Graphs can be classified by whether or not their edges have weights
- Edges simply show connections Adjacency Matrix: 2D array containing weights on edges
- Row for each vertex
- Column for each vertex
- Entries contain weight of edge from row vertex to column vertex
- Entries contain ∞ (ie Integer'last) if no edge from row vertex to column vertex
- Entries contain 0 on diagonal (if self edges not allowed)

undirected graph

	A	B	C	D
A	0	1	1	1
B	1	0	∞	1
C	1	∞	0	1
D	1	1	1	0



Applications of Graphs

- Social network graphs: to tweet or not to tweet
- Transportation networks
- Utility graphs
- Document link graphs
- Protein-protein interactions graphs
- Network packet traffic graphs
- Scene graphs
- Finite element meshes.
- Neural networks
- Robot planning

Video Content / Details of Itbsite for further learning (if any):

http://www.btechsmartclass.com/data_structures/graph-representations.html

https://nptel.ac.in/courses/106/106/106106133/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 300-301

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L-33**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Breadth First Search

Introduction :

- Graph traversal is a technique used for a searching vertex in a graph.
- The graph traversal is also used to decide the order of vertices is visited in the search process.
- A graph traversal finds the edges to be used in the search process without creating loops.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Two important topics)

- Queue
- Vertex
- Edges
- Graph traversal

Detailed content of the Lecture:

BFS (Breadth First Search)

- BFS traversal of a graph produces a **spanning tree** as final result.
- **Spanning Tree** is a graph without loops.
- BFS use **Queue data structure** with maximum size of total number of vertices in the graph to implement BFS traversal.

Steps to implement BFS traversal.

Step 1 - Define a Queue of size total number of vertices in the graph.

Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.

Step 3 - Visit all the non-visited **adjacent** vertices of the vertex which is at front of the Queue and insert them into the Queue.

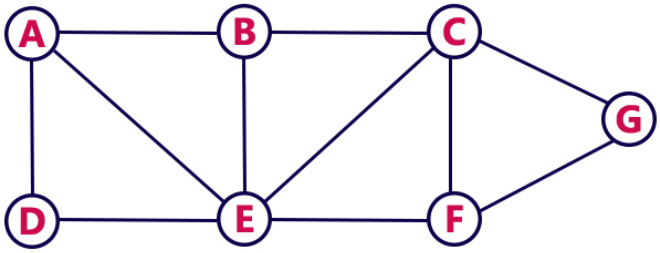
Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.

Step 5 - Repeat steps 3 and 4 until queue becomes empty.

Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

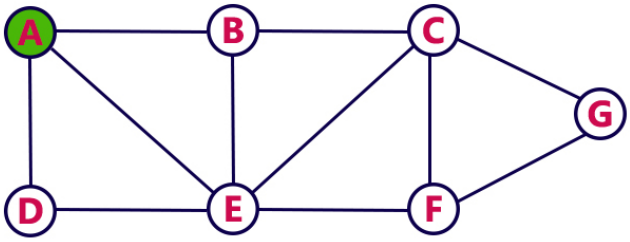
EXAMPLE

Consider the following example graph to perform BFS traversal



Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.

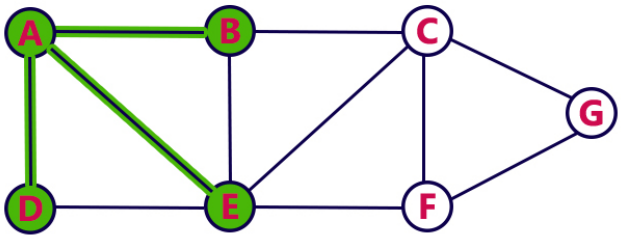


Queue

A						

Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..

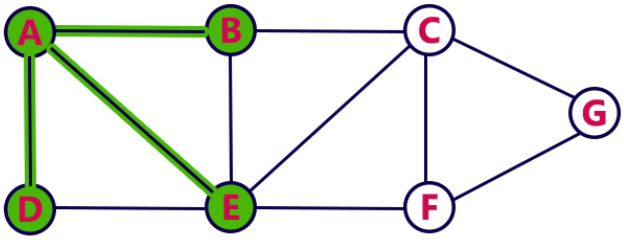


Queue

	D	E	B			

Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.

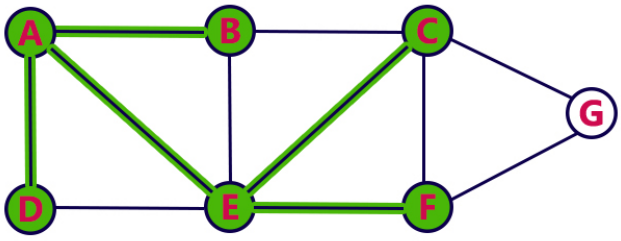


Queue

		E	B			

Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.

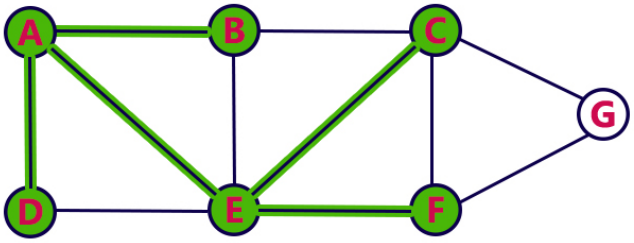


Queue

			B	C	F	

Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.

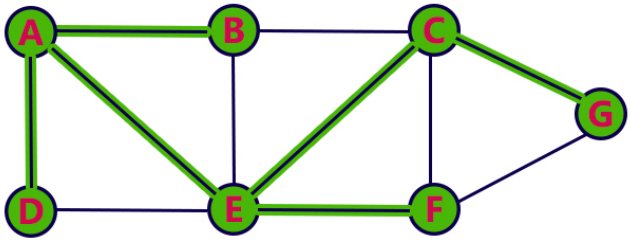


Queue

				C	F	

Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.

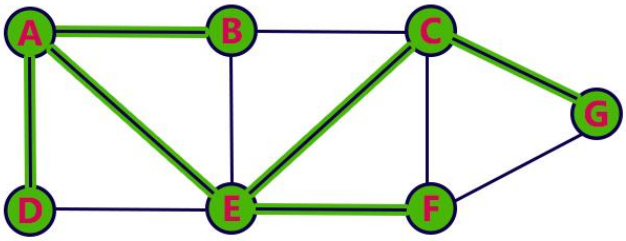


Queue

					F	G

Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.

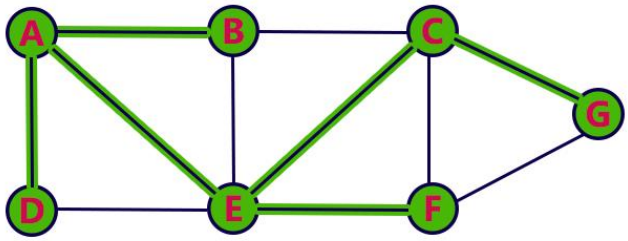


Queue

						G

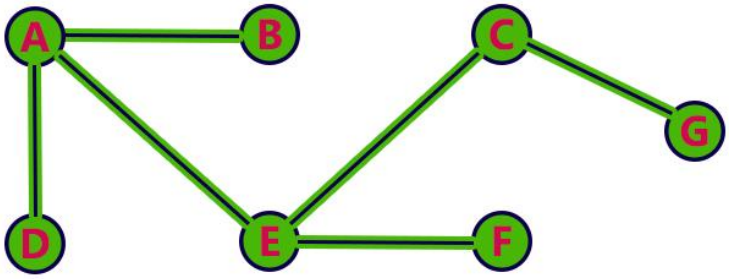
Step 8:

- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.



Queue

- Queue became Empty. So, stop the BFS process.
- Final result of BFS is a Spanning Tree as shown below...



Video Content / Details of website for further learning (if any):
http://www.btechsmartclass.com/data_structures/graph-traversal-bfs.html

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:**335-336

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L-34**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Depth First Search

Introduction :

- Graph traversal is a technique used for a searching vertex in a graph.
- The graph traversal is also used to decide the order of vertices is visited in the search process.
- A graph traversal finds the edges to be used in the search process without creating loops.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Two important topics)

- Stack
- Vertex
- Edges
- Graph traversal

Detailed content of the Lecture:

DFS (Depth First Search)

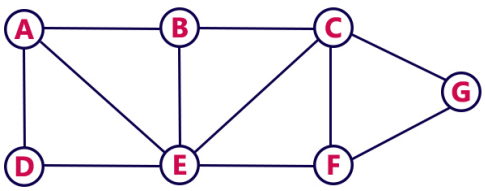
- DFS traversal of a graph produces a **spanning tree** as final result.
- **Spanning Tree** is a graph without loops.
- DFS use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal.

Steps to implement DFS traversal.

step 1 - Define a Stack of size total number of vertices in the graph.
Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.
Step 3 - Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of stack and push it on to the stack.
Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
Step 5 - When there is no new vertex to visit then use **back tracking** and pop one vertex from the stack.
Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.
Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

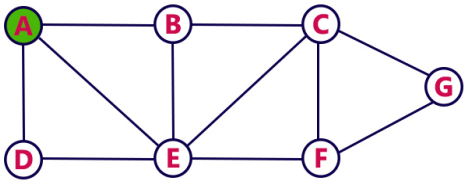
Example

Consider the following example graph to perform DFS traversal



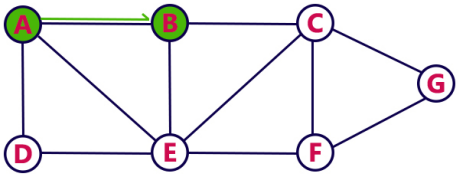
Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.



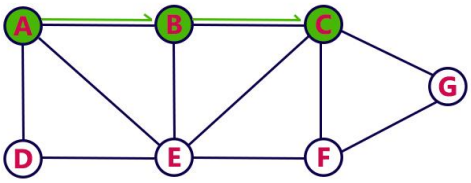
Step 2:

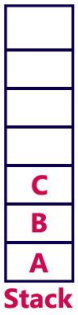
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.



Step 3:

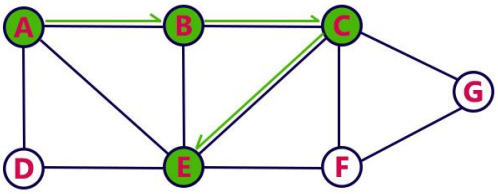
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.





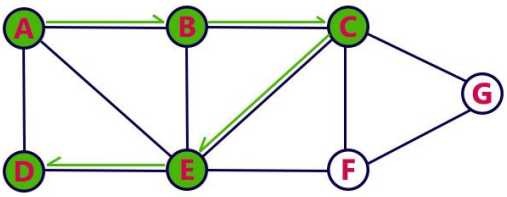
Step 4:

- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack



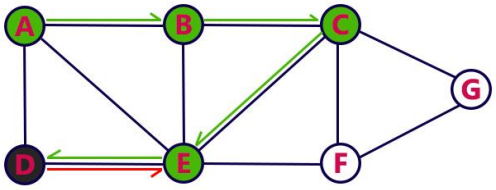
Step 5:

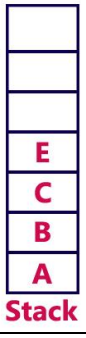
- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack



Step 6:

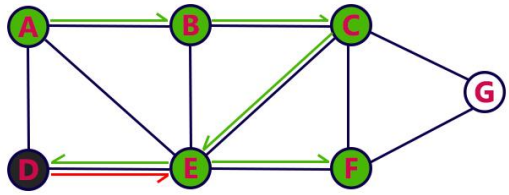
- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.

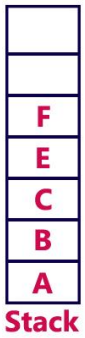




Step 7:

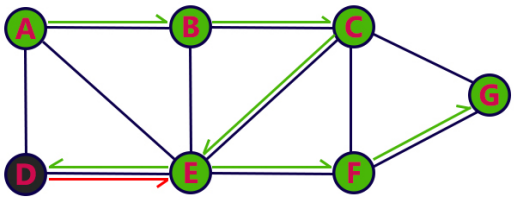
- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.

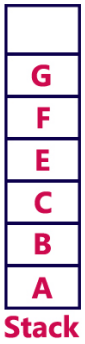




Step 8:

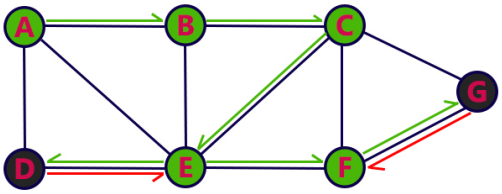
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.





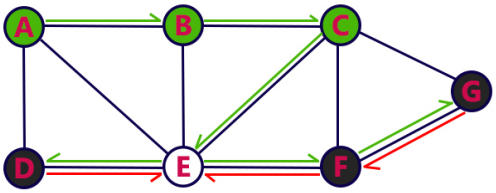
Step 9:

- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.



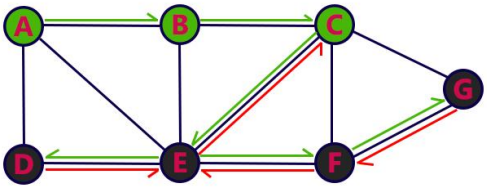
Step 10:

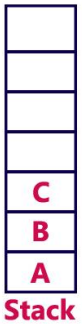
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.



Step 11:

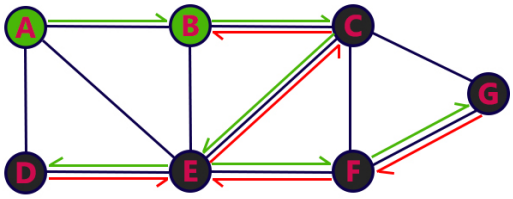
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.





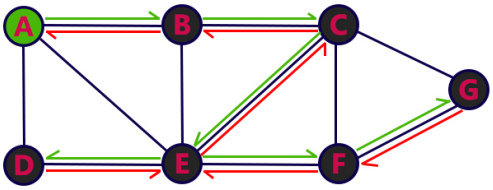
Step 12:

- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.



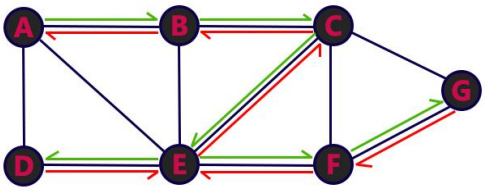
Step 13:

- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.

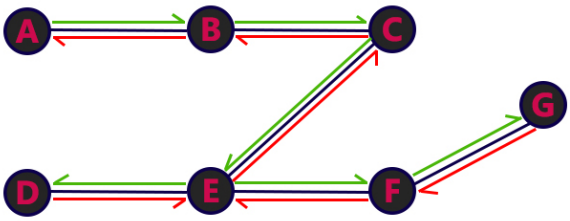


Step 14:

- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.



- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.



Video Content / Details of website for further learning (if any):
http://www.btechsmartclass.com/data_structures/graph-traversal-dfs.html

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 333-334

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L35**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Spanning Trees, Minimal spanning tree

Introduction :

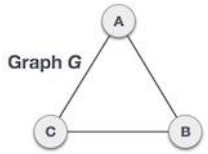
- A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges.
- Hence, a spanning tree does not have cycles and it cannot be disconnected.

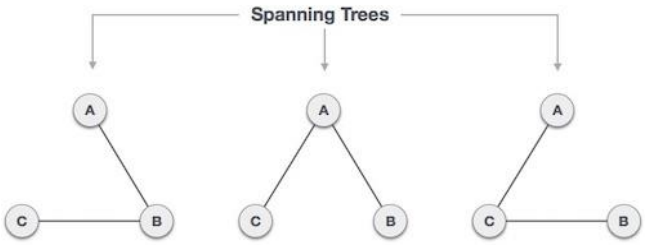
Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Two important topics)

- vertices
- Graphs
- Nodes
- cycles

Detailed content of the Lecture:

- Given a graph G=(V,E), a subgraph of G that is connects all of the vertices and is a **tree** is called a **spanning tree** .
- For **example**, suppose we start with this graph: We can remove edges until we are left with a **tree**: the result is a **spanning tree**.
- Clearly, a **spanning tree** will have |V|-1 edges, like any other **tree**.





three spanning trees off one complete graph.

A complete undirected graph can have maximum $\mathbf{n^{n-2}}$ number of spanning trees, where **n** is the number of nodes. In the above addressed example, **n is 3,** hence $\mathbf{3^{3-2} = 3}$ spanning trees are possible.

General Properties of Spanning Tree

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning

tree is **minimally connected**.

- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is **maximally acyclic**.

Mathematical Properties of Spanning Tree

- Spanning tree has **n-1** edges, where **n** is the number of nodes (vertices).
- From a complete graph, by removing maximum **e - n + 1** edges, we can construct a spanning tree.
- A complete graph can have maximum $\mathbf{n^{n-2}}$ number of spanning trees.
- Thus, we can conclude that spanning trees are a subset of connected Graph G and disconnected graphs do not have spanning tree.

Application of Spanning Tree

- Spanning tree is basically used to find a minimum path to connect all nodes in a graph.

Common application of spanning trees are −

- Civil Network Planning
- Computer Network Routing Protocol
- Cluster Analysis

Video Content / Details of website for further learning (if any):
https://www.youtube.com/watch?v=ImSRt7LxQnY

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 306-320

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS | **L36**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : IV [GRAPHS] **Date of Lecture:**

Topic of Lecture: Minimal spanning tree & Hamiltonian circuit

Introduction :

- Connected (there exists a path between every pair of vertices)
- Undirected (the edges do no have any directions associated with them such that (a,b) and (b,a) are equivalent)
- Weighted (each edge has a weight or cost assigned to it)
- A **Hamiltonian cycle** (or **Hamiltonian circuit**) is a **Hamiltonian Path** such that there is an edge (in the graph) from the last vertex to the first vertex of the **Hamiltonian Path**. ... A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph.

Prerequisite knowledge for Complete understanding and learning of Topic:

- spanning tree
- cycle
- acyclic
- Hamiltonian cycle
- Hamiltonian circuit
- Hamiltonian Path

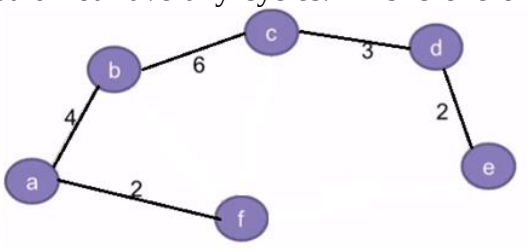
Detailed content of the Lecture:

The network shown in the second figure basically represents a graph **G = (V, E)** with a set of vertices **V = {a, b, c, d, e, f}** and a set of edges **E = { (a,b), (b,c), (c,d), (d,e), (e,f), (f,a), (b,f), (c,f) }**. The graph is:

- Connected (there exists a path between every pair of vertices)
- Undirected (the edges do no have any directions associated with them such that (a,b) and (b,a) are equivalent)
- Weighted (each edge has a weight or cost assigned to it)

A spanning tree G' = (V, E') for the given graph G will include:

- All the vertices (V) of G
- All the vertices should be connected by minimum number of edges (E') such that E' ⊂ E
- G' can have maximum n-1 edges, where n is equal to the total number of edges in G
- G' should not have any cycles. This is one of the basic differences between a tree and graph



that **a** Figure 2

Also, there can be multiple spanning trees possible for any given graph. For eg: In addition to the

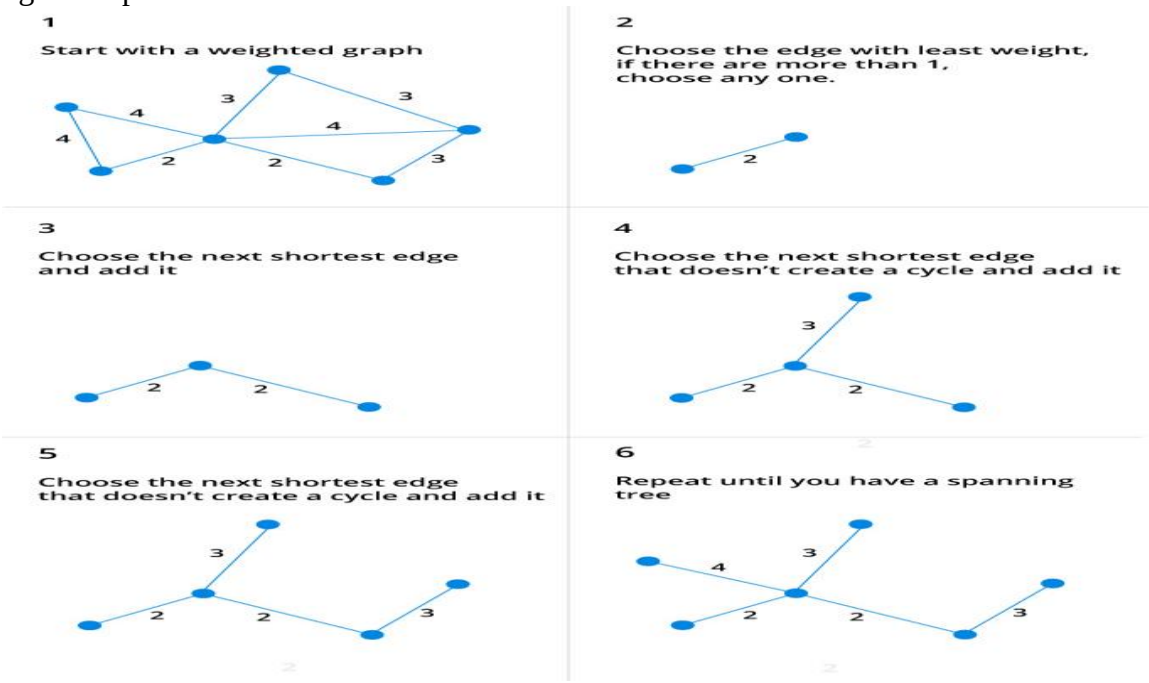
spanning tree in the above diagram, the graph can also have another spanning tree as shown below:
By convention, the total number of spanning trees for a given graph can be defined as:
${}^{n}C_{m}$ = n!/(m!*(n-m)!), where,
Hence, the total number of spanning trees(S) for the given graph(second diagram from top) can be computed as follows:

- **n = 8**, for the given graph in Fig. 2
- **m = 5**, since its corresponding spanning tree can have only 5 edges. Adding a 6th edge can result in the formation of cycles which is not allowed.
- So, **$S = {}^{n}C_{m} = {}^{8}C_{5} = 8!/ (5! * 3!) = 56$**, which means that **56** different variations of spanning trees can be created for the given graph.

Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which form a tree that includes every vertex has the minimum sum of weights among all the trees that can be formed from the graph

How Kruskal's algorithm works

It falls under a class of algorithms called greedy algorithms which find the local optimum in the hopes of finding a global optimum.



Kruskal's

Kruskal Algorithm Pseudocode

Any minimum spanning tree algorithm revolves around checking if adding an edge creates a loop or not.

```
KRUSKAL(G):
A = ∅
For each vertex v ∈ G.V:
   MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
   if FIND-SET(u) ≠ FIND-SET(v):
   A = A ∪ {(u, v)}
   UNION(u, v)
return A
```

A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path. ... A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph.

Given a graph G = (V, E) we have to find the Hamiltonian Circuit using Backtracking approach. We start our search from any arbitrary vertex say 'a.'

This vertex 'a' becomes the root of our implicit tree. The first element of our partial solution is the first intermediate vertex of the Hamiltonian Cycle that is to be constructed.

The next adjacent vertex is selected by alphabetical order. If at any stage any arbitrary vertex makes a cycle with any vertex other than vertex 'a' then we say that dead end is reached.
In this case, we backtrack one step, and again the search begins by selecting another vertex and backtrack the element from the partial; solution must be removed.
The search using backtracking is successful if a Hamiltonian Cycle is obtained.
In an undirected graph, the Hamiltonian path is a path, that visits each vertex exactly once, and the Hamiltonian cycle or circuit is a Hamiltonian path, that there is an edge from the last vertex to the first vertex.Algorithm

isValid(v, k)

Input: Vertex v and position k.

Output: Checks whether placing v in the position k is valid or not.

```
Begin
  if there is no edge between node(k-1) to v, then
    return false
  if v is already taken, then
    return false
  return true; //otherwise it is valid
End
```

cycleFound(node k)

Input: node of the graph.

Output: True when there is a Hamiltonian Cycle, otherwise false.

```
Begin
  if all nodes are included, then
    if there is an edge between nodes k and 0, then
      return true
    else
      return false;

  for all vertex v except starting point, do
    if isValid(v, k), then //when v is a valid edge
      add v into the path
      if cycleFound(k+1) is true, then
        return true
      otherwise remove v from the path
  done
  return false
End
```

Source Code (C++)

```
#include<iostream>
#define NODE 5
using namespace std;

int graph[NODE][NODE] = {
  {0, 1, 0, 1, 0},
  {1, 0, 1, 1, 1},
  {0, 1, 0, 0, 1},
  {1, 1, 0, 0, 1},
  {0, 1, 1, 1, 0},
};

int path[NODE];

void displayCycle() {
  cout<<"Cycle: ";

  for (int i = 0; i < NODE; i++)
    cout << path[i] << " ";
  cout << path[0] << endl;      //print the first vertex again
          }
```

Video Content / Details of website for further learning (if any):
https://nptel.ac.in/courses/106/106/106106127/

Important Books/Journals for further learning including the page nos.:
R.Kruse,C.L.Tondo and B.Leung,Data structures and Program Design in C, 2nd Edition , Prentice-Hall, 2006,**page nos:** 33-35,332

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L37**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Hashing: Introduction, Hash table, Hash function, Collision, Collision resolution

Introduction

- Hashing is also known as Hashing Algorithm or Message Digest Function..
- Function which helps us in generating such kind of key-value mapping is known as Hash Function.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Array
- Index value
- key-value mapping

Detailed content of the Lecture:

- **Hashing** is the process of mapping large amount of **data** item to smaller table with the help of **hashing** function.
- It is a technique to convert a range of key values into a range of indexes of an array.
- It is used to facilitate the next level searching method when compared with the linear or binary search.
- Hashing allows to update and retrieve any data entry in a constant time O(1).
- Constant time O(1) means the operation does not depend on the size of the data.
- Hashing is used with a database to enable items to be retrieved more quickly.
- It is used in the encryption and decryption of digital signatures.

Hash Function

- A fixed process converts a key to a hash key is known as a **Hash Function.**
- This function takes a key and maps it to a value of a certain length which is called a **Hash value** or **Hash.**
- Hash value represents the original string of characters, but it is normally smaller than the original.
- It transfers the digital signature and then both hash value and signature are sent to the receiver. Receiver uses the same hash function to generate the hash value and then compares it to that received with the message.
- If the hash values are same, the message is transmitted without errors.

Hash Table

- Hash table or hash map is a data structure used to store key-value pairs.
- It is a collection of items stored to make it easy to find them later.

- It uses a hash function to compute an index into an array of buckets or slots from which the desired value can be found.
- It is an array of list where each list is known as bucket.
- It contains value based on the key.
- Hash table is used to implement the map interface and extends Dictionary class.
- Hash table is synchronized and contains only unique elements.

Suppose we have integer items {26, 70, 18, 31, 54, 93}. One common method of determining a hash key is the division method of hashing and the formula is :

Hash Key = Key Value % Number of Slots in the Table

Division method or reminder method takes an item and divides it by the table size and returns the remainder as its hash value.

Data Item	Value % No. of Slots	Hash Value
26	26 % 10 = 6	6
70	70 % 10 = 0	0
18	18 % 10 = 8	8
31	31 % 10 = 1	1
54	54 % 10 = 4	4
93	93 % 10 = 3	3

Fig. Hash Table

Collision Handling:

- Since a hash function gets us a small number for a big key, there is possibility that two keys result in same value.
- The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique.
- Following are the ways to handle collisions:
  - **Chaining:**The idea is to make each cell of hash table point to a linked list of records that have same hash function value.
  - **Open Addressing:** In open addressing, all elements are stored in the hash table itself.

Video Content / Details of website for further learning (if any):
https://www.tutorialride.com/data-structures/hashing-in-data-structure.html
https://nptel.ac.in/courses/106105085/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 165-167

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L38**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Separate chaining, open addressing; Rehashing – Extendible hashing

Introduction :

- It is a situation in which the hash function returns the same hash key for more than one record, it is called as collision.
- Sometimes when we are going to resolve the collision it may lead to a overflow condition and this overflow and collision condition makes the poor hash function.
- It is a dynamic hashing method wherein directories, and buckets are used to hash data.
- It is an aggressively flexible method in which the hash function also experiences dynamic changes.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Hashing
- Hash table
- Collision

Detailed content of the Lecture:

Collision resolution technique

- If there is a problem of collision occurs then it can be handled by apply some technique. These techniques are called as collision resolution techniques.
- There are generally four techniques which are described below.

Chaining

- It is a method in which additional field with data i.e. chain is introduced.
- A chain is maintained at the home bucket.
- In this when a collision occurs then a linked list is maintained for colliding data.

Open Addressing

- Like separate chaining, open addressing is a method for handling collisions.
- In Open Addressing, all elements are stored in the hash table itself.
- So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).
- Insert(k): Keep probing until an empty slot is found.

- Once an empty slot is found, insert k.
- Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.
- Delete(k): ***Delete operation is interesting***. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".
  Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Linear probing

- It is very easy and simple method to resolve or to handle the collision.
- In this collision can be solved by placing the second record linearly down, whenever the empty place is found.
- In this method there is a problem of clustering which means at some place block of a data is formed in a hash table.

Quadratic probing

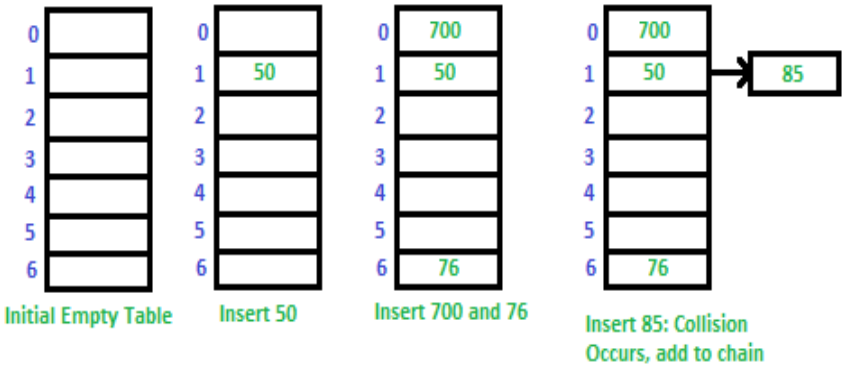
- This is a method in which solving of clustering problem is done.
- In this method the hash function is defined by the H(key)=(H(key)+x*x)%table size.

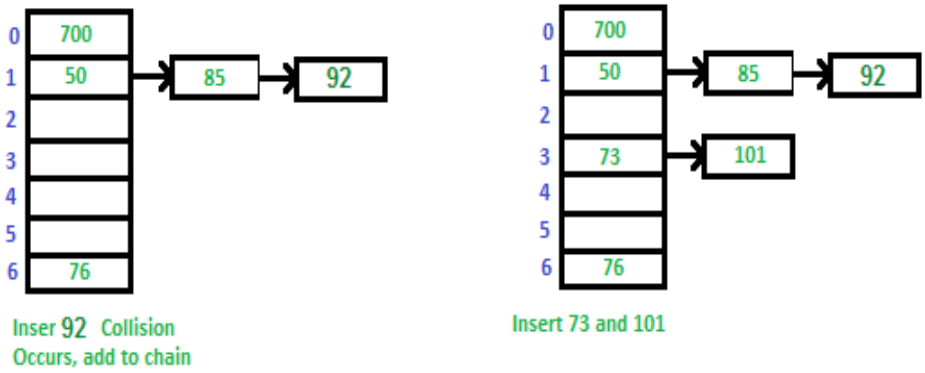
Double hashing

- It is a technique in which two hash function are used when there is an occurrence of collision.
- It must never evaluate to zero.
- Must sure about the buckets, that they are probed.
- The hash functions for this technique are:
  - H1(key)=key % table size
  - H2(key)=P-(key mod P)
- Where, **p** is a prime number which should be taken smaller than the size of a hash table.

Separate Chaining:

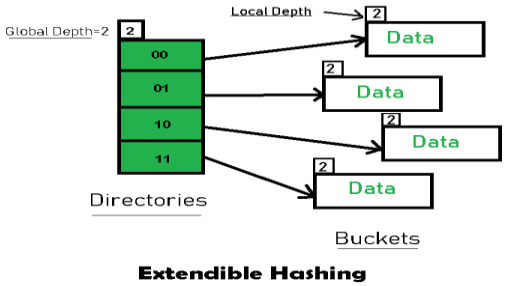
- The idea is to make each cell of hash table point to a linked list of records that have same hash function value.
- Let us consider a simple hash function as "**key mod 7**" and sequence of keys as 50, 700, 76, 85, 92, 73, 101.





Main features of Extendible Hashing:

- The main features in this hashing technique are:
- Directories: The directories store addresses of the buckets in pointers. An id is assigned to each directory which may change each time when Directory Expansion takes place.
- Buckets: The buckets are used to hash the actual data.



Extendible Hashing

Frequently used terms in Extendible Hashing:

- **Directories:** These containers store pointers to buckets. Each directory is given a unique id which may change each time when expansion takes place. The hash function returns this directory id which is used to navigate to the appropriate bucket. Number of Directories = 2^Global Depth.

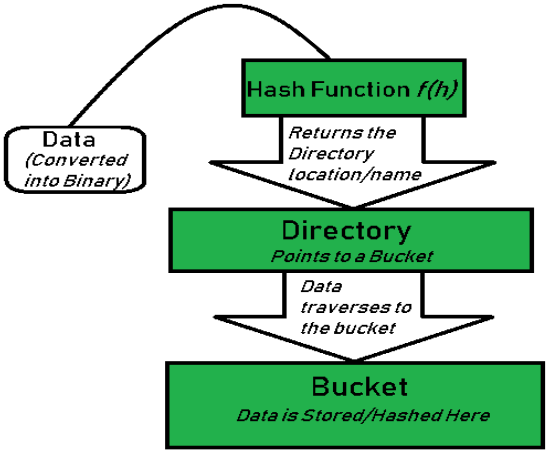
Buckets: They store the hashed keys. Directories point to buckets. A bucket may contain more than one pointers to it if its local depth is less than the global depth.

Global Depth: It is associated with the Directories. They denote the number of bits which are used by the hash function to categorize the keys. Global Depth = Number of bits in directory id.

Local Depth: It is the same as that of Global Depth except for the fact that Local Depth is associated with the buckets and not the directories. **Bucket Splitting:** When the number of elements in a bucket exceeds a particular size, then the bucket is split into two parts.

Directory Expansion: Directory Expansion Takes place when a bucket overflows. Directory Expansion is performed when the local depth of the overflowing bucket is equal to the global depth.

Basic Working of Extendible Hashing:



Step 1 – Analyze Data Elements: Data elements may exist in various forms eg. Integer, String, Float, etc.. Currently, let us consider data elements of type integer. eg: 49.

Step 2 – Convert into binary format: Convert the data element in Binary form. For string elements, consider the ASCII equivalent integer of the starting character and then convert the integer into binary form. Since we have 49 as our data element, its binary form is 110001.

Step 3 – Check Global Depth of the directory. Suppose the global depth of the Hash-directory is 3.

Step 4 – Identify the Directory: Consider the 'Global-Depth' number of LSBs in the binary number and match it to the directory id.

Eg. The binary obtained is: 110001 and the global-depth is 3. So, the hash function will return 3 LSBs of 110**001** viz. 001.

Step 5 – Navigation: Now, navigate to the bucket pointed by the directory with directory-id 001.

Step 6 – Insertion and Overflow Check: Insert the element and check if the bucket overflows. If an overflow is encountered, go to **step 7** followed by **Step 8**, otherwise, go to **step 9**.

Step 7 – Tackling Over Flow Condition during Data Insertion: Many times, while inserting data in the buckets, it might happen that the Bucket overflows. In such cases, we need to follow an appropriate procedure to avoid mishandling of data.

Advantages:

1. Data retrieval is less expensive (in terms of computing).
2. No problem of Data-loss since the storage capacity increases dynamically.
3. With dynamic changes in hashing function, associated old values are rehashed w.r.t the new hash function.

Limitations Of Extendible Hashing:

1. The directory size may increase significantly if several records are hashed on the same directory while keeping the record distribution non-uniform.
2. Size of every bucket is fixed.
3. Memory is wasted in pointers when the global depth and local depth difference becomes drastic.

This method is complicated to code.

Video Content / Details of Itbsite for further learning (if any):

https://www.includehelp.com/data-structure-tutorial/hashing.aspx

https://www.youtube.com/watch?v=Z-eW5qp7lvk

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 167-180

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L39**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Searching: Definition – Algorithm and Example for sequential search

Introduction :

- Searching is a process of finding a particular data item from a collection of data items based on specific criteria.
- Searching is the process of locating given value position in a list of values.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Data structure
- Searching
- Index value

Detailed content of the Lecture:

Searching

- Searching is the process of finding a given value position in a list of values.
- It decides whether a search key is present in the data or not.
- It is the algorithmic process of finding a particular item in a collection of items.
- It can be done on internal data structure or on external data structure.

Searching Techniques

To search an element in a given array, it can be done in following ways:

1. Sequential Search

2. Binary Search

Sequential Search

- Sequential search is also called as Linear Search.
- Sequential search starts at the beginning of the list and checks every element of the list.
- It is a basic and simple search algorithm.
- Sequential search compares the element with all the other elements given in the list. If the element is matched, it returns the value index, else it returns -1.

Algorithm

Linear search is implemented using following steps.

Step 1 - Read the search element from the user.

Step 2 - Compare the search element with the first element in the list.

Step 3 - If both are matched, then display "Given element is found!!!" and terminate the function

Step 4 - If both are not matched, then compare search element with the next element in the list.

Step 5 - Repeat steps 3 and 4 until search element is compared with last element in the list.

Step 6 - If last element in the list also doesn't match, then display "Element is not found!!!" and terminate the function.

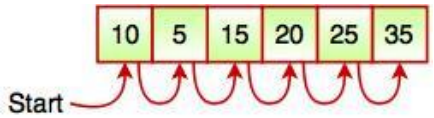


Fig. Sequential Search

- It searches an element or value from an array till the desired element or value is not found.
- If we search the element 25, it will go step by step in a sequence order.
- It searches in a sequence order.
- Sequential search is applied on the unsorted or unordered list when there are fewer elements in a list.

Pseudocode

```
procedure linear_search (list, value)

   for each item in the list
      if match item == value
         return the item's location
      end if
   end for
end procedure
```

Video Content / Details of Itbsite for further learning (if any):

http://www.btechsmartclass.com/data_structures/linear-search.html

https://www.youtube.com/watch?v=Z-eW5qp7lvk

Important Books/Journals for further learning including the page nos.:

E.Horowitz,S.Sahni Susan , Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008- **page nos:** 65-68

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L40**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Binary search.

Introduction :

- Searching is a process of finding a particular data item from a collection of data items based on specific criteria.
- Searching is the process of locating given value position in a list of values.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Data Structure
- Searching
- Index Value

Detailed content of the Lecture:

- Binary Search is used for searching an element in a sorted array.
- It is a fast search algorithm with run-time complexity of O(log n).
- Binary search works on the principle of divide and conquer.
- This searching technique looks for a particular element by comparing the middle most element of the collection.
- It is useful when there are large number of elements in an array.

- The above array is sorted in ascending order. As we know binary search is applied on sorted lists only for fast searching.

Algorithm

Binary search is implemented using following steps.

Step 1 - Read the search element from the user.
Step 2 - Find the middle element in the sorted list.
Step 3 - Compare the search element with the middle element in the sorted list.
Step 4 - If both are matched, then display "Given element is found!!!" and terminate the function.
Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element.

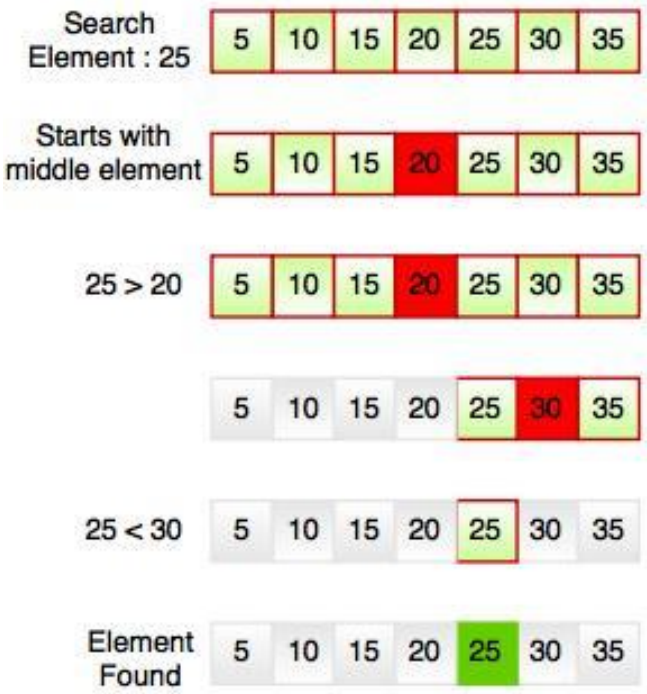
Step 6 - If the search element is smaller than middle element, repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.

Step 7 - If the search element is larger than middle element, repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.

Step 8 - Repeat the same process until we find the search element in the list or until sublist contains only one element.

Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function.

For example, if searching an element 25 in the 7-element array, following figure shows how binary search works:



Video Content / Details of Itbsite for further learning (if any):

http://www.btechsmartclass.com/data_structures/binary-search.html

https://www.youtube.com/watch?v=Z-eW5qp7lvk

Important Books/Journals for further learning including the page nos.:

E.Horowitz,S.Sahni Susan , Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008- **page nos:**69-72

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L41**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Sorting: Definition – Algorithm and Example for selection sort

Introduction :

- Sorting is the process of arranging a list of elements in a particular order (Ascending or Descending).

Prerequisite knowledge for Complete understanding and learning of Topic:

- Array
- Index value
- sorting

Detailed content of the Lecture:

Sorting

- Sorting is a process of ordering or placing a list of elements from a collection in some kind of order.
- It is nothing but storage of data in sorted order.
- Sorting can be done in ascending and descending order.
- It arranges the data in a sequence which makes searching easier.

Selection Sort

- Selection sort is a simple sorting algorithm which finds the smallest element in the array and exchanges it with the element in the first position.
- Then finds the second smallest element and exchanges it with the element in the second position and continues until the entire array is sorted.

Algorithm

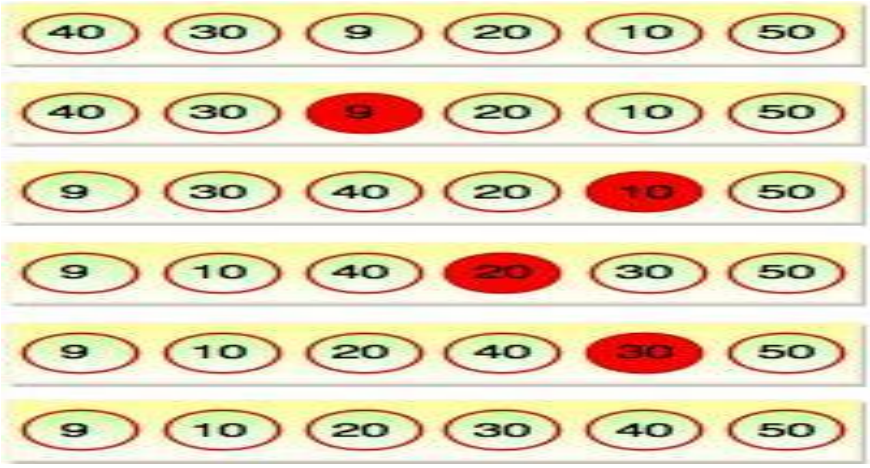
Step 1 - Select the first element of the list (i.e., Element at first position in the list).

Step 2: Compare the selected element with all the other elements in the list.

Step 3: In every comparision, if any element is found smaller than the selected element (for Ascending order), then both are swapped.

Step 4: Repeat the same procedure with element in the next position in the list till the entire list is sorted.

Example



- The smallest element is found in first pass that is 9 and it is placed at the first position.
- In second pass, smallest element is searched from the rest of the element excluding first element.Selection sort keeps doing this, until the array is sorted.

Video Content / Details of Itbsite for further learning (if any):
https://www.tutorialride.com/data-structures/selection-sort-in-data-structure.htm
https://www.youtube.com/watch?v=4OxBvBXon5w

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **Page Nos:** 235-248

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L42**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Bubble sort, Insertion sort

Introduction :

- Sorting is the process of arranging a list of elements in a particular order (Ascending or Descending).

Prerequisite knowledge for Complete understanding and learning of Topic:

- Array
- Index value
- sorting

Detailed content of the Lecture:

Sorting

- Sorting is a process of ordering or placing a list of elements from a collection in some kind of order.
- It is nothing but storage of data in sorted order.
- Sorting can be done in ascending and descending order.
- It arranges the data in a sequence which makes searching easier.

Bubble sort

- Bubble sort is a type of sorting.
- It is used for sorting 'n' (number of items) elements.
- It compares all the elements one by one and sorts them based on their values.

Algorithm

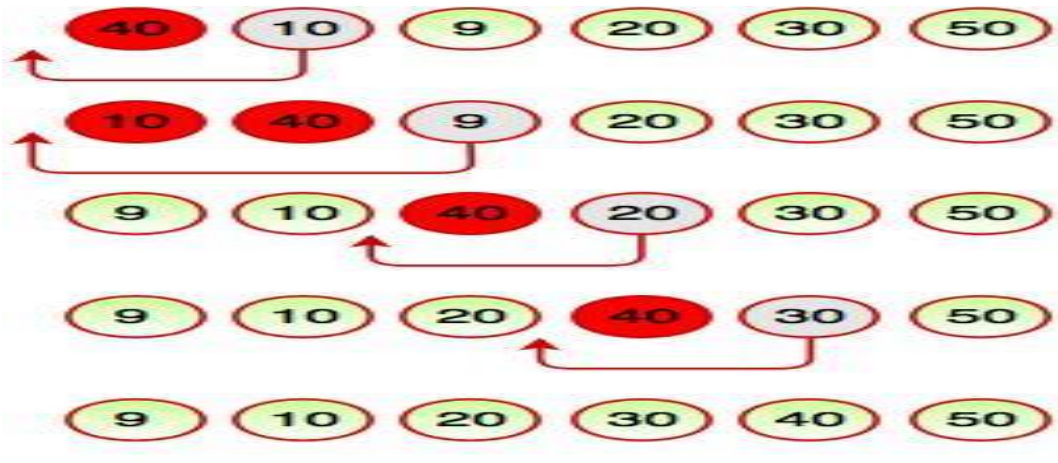
- Starting with the first element(index = 0), compare the current element with the next element of the array.
- If the current element is greater than the next element of the array, swap them.
- If the current element is less than the next element, move to the next element. Repeat Step 1.

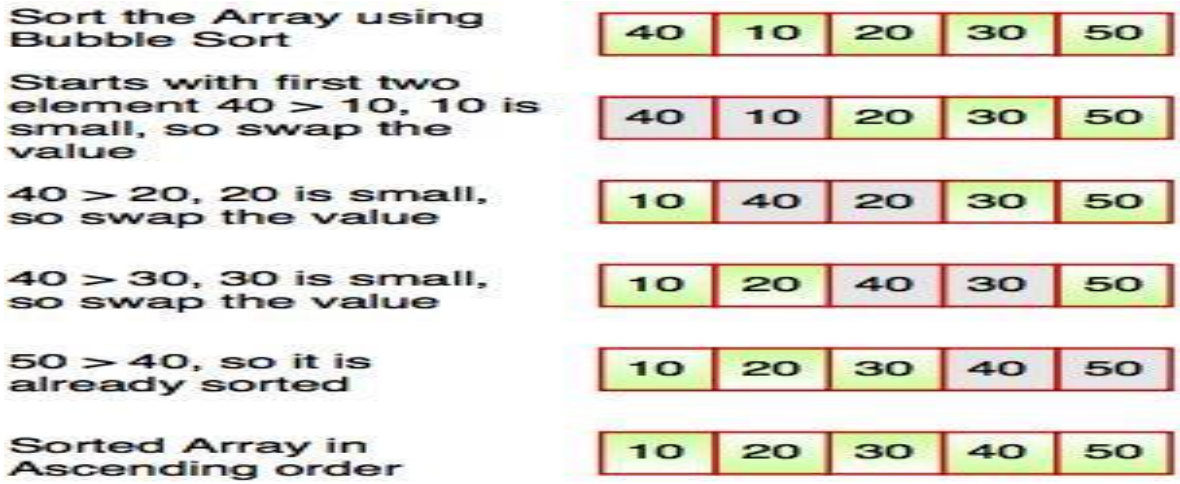
Insertion Sort

- Insertion sort is a simple sorting algorithm.
- This sorting method sorts the array by shifting elements one by one.
- It builds the final sorted array one item at a time.
- Insertion sort has one of the simplest implementation.
- This sort is efficient for smaller data sets but it is insufficient for larger lists.
- It has less space complexity like bubble sort.
- It requires single additional memory space.
- Insertion sort does not change the relative order of elements with equal keys because it is stable.

Algorithm

Step 1 − If it is the first element, it is already sorted. return 1;
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list
Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 − Insert the value
Step 6 − Repeat until list is sorted





-

Video Content / Details of Itbsite for further learning (if any):
https://www.tutorialride.com/data-structures/selection-sort-in-data-structure.htm
https://www.youtube.com/watch?v=4OxBvBXon5w

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 235-248

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS | **L43**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Quick sort, merge sort

Introduction :

- Sorting is the process of arranging a list of elements in a particular order (Ascending or Descending).

Prerequisite knowledge for Complete understanding and learning of Topic:

- Pivot
- Index value
- sorting

Detailed content of the Lecture:

Quick Sort

- Quick sort is also known as Partition-exchange sort based on the rule of Divide and Conquer.
- It is a highly efficient sorting algorithm.
- Quick sort is the quickest comparison-based sorting algorithm.
- It is very fast and requires less additional space, only O(n log n) space is required.
- Quick sort picks an element as pivot and partitions the array around the picked pivot.

Algorithm for Quick Sort

Step 1: Choose the highest index value as pivot.

Step 2: Take two variables to point left and right of the list excluding pivot.

Step 3: Left points to the low index.

Step 4: Right points to the high index.

Step 5: While value at left < (Less than) pivot move right.

Step 6: While value at right > (Greater than) pivot move left.

Step 7: If both Step 5 and Step 6 does not match, swap left and right.

Step 8: If left = (Less than or Equal to) right, the point where they met is new pivot.

Example

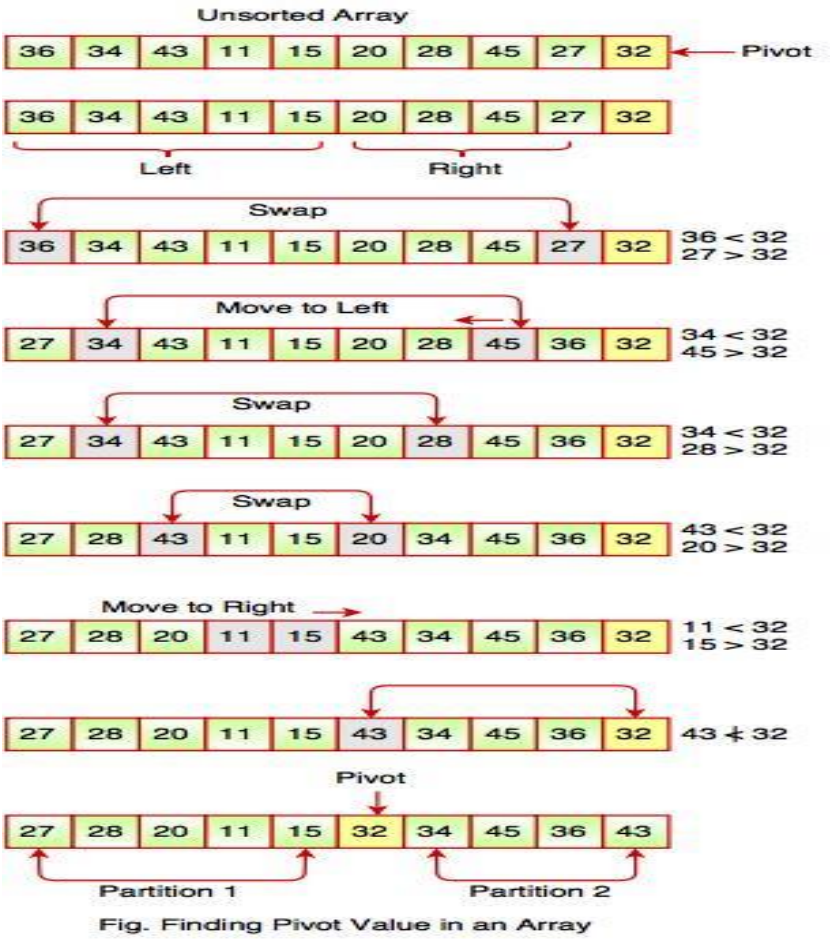


Fig. Finding Pivot Value in an Array

Merge Sort

- **Merge sort** is a **sorting** technique based on divide and conquer technique.
- With worst-case time complexity being O(n log n), it is one of the most respected **algorithms**.
- **Merge sort** first divides the array into equal halves and then combines them in a **sorted** manner.

The concept of Divide and Conquer involves three steps:

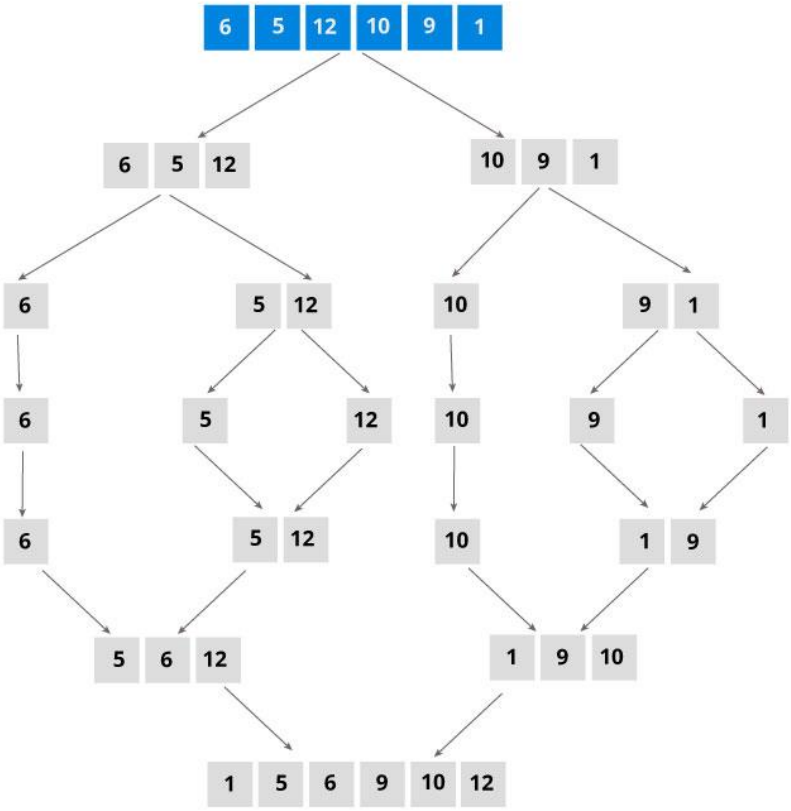
- **Divide** the problem into multiple small problems.
- **Conquer** the subproblems by solving them. The idea is to break down the problem into atomic subproblems, where they are actually solved.
- **Combine** the solutions of the subproblems to find the solution of the actual problem.

Algorithm

Step 1 − if it is only one element in the list it is already sorted, return.

Step 2 − divide the list recursively into two halves until it can no more be divided.

Step 3 − merge the smaller lists into new list in sorted order.



Video Content / Details of Itbsite for further learning (if any):
https://www.tutorialspoint.com/data_structures_algorithms/insertion_sort_algorithm.htm
https://nptel.ac.in/courses/106105164/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 248-255

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS	L44

IT	II/III

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Radix sort

Introduction :

- Sorting is the process of arranging a list of elements in a particular order (Ascending or Descending).

Prerequisite knowledge for Complete understanding and learning of Topic:

- Pivot
- Index value
- sorting

Detailed content of the Lecture:

Radix sort

- Radix sort is one of the sorting algorithms used to sort a list of integer numbers in order.
- In radix sort algorithm, a list of integer numbers will be sorted based on the digits of individual numbers.
- Sorting is performed from **least significant digit to the most significant digit**.
  Radix sort algorithm requires the number of passes which are equal to the number of digits present in the largest number among the list of numbers.

Algorithm

Step 1 - Define 10 queues each representing a bucket for each digit from 0 to 9.
Step 2 - Consider the least significant digit of each number in the list which is to be sorted.
Step 3 - Insert each number into their respective queue based on the least significant digit.
Step 4 - Group all the numbers from queue 0 to queue 9 in the order they have inserted into their respective queues.
Step 5 - Repeat from step 3 based on the next least significant digit.
Step 6 - Repeat from step 2 until all the numbers are grouped based on the most significant digit.

Complexity of the Radix sort

- Worst Case : O(n)
- Best Case : O(n)
- Average Case : O(n)

Video Content / Details of Itbsite for further learning (if any):

http://www.btechsmartclass.com/data_structures/radix-sort.html
https://nptel.ac.in/courses/106105164/

Important Books/Journals for further learning including the page nos.:

Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 256-261

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L45**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Heap Sort

Introduction :

- Sorting is the process of arranging a list of elements in a particular order (Ascending or Descending).

Prerequisite knowledge for Complete understanding and learning of Topic:

- Pivot
- Index value
- sorting

Detailed content of the Lecture:

Heap Sort

- Heap sort is a comparison based sorting algorithm.
- It is a special tree-based data structure.
- Heap sort is similar to selection sort. The only difference is, it finds largest element and places the it at the end.
- This sort is not a stable sort. It requires a constant space for sorting a list.
- It is very fast and widely used for sorting.

It has following two properties:

1. Structure Property

2. Heap order Property

1. Shape property represents all the nodes or levels of the tree are fully filled. Heap data structure is a complete binary tree.

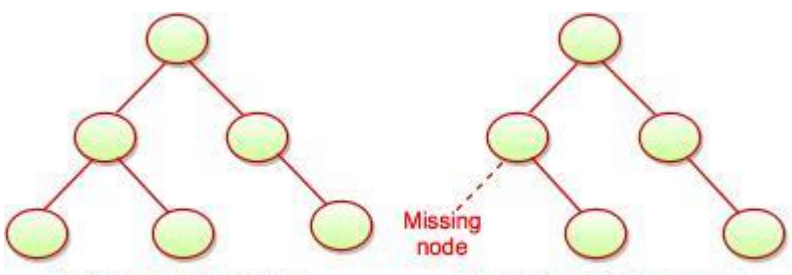


Fig. Complete-Binary Tree Fig. In-Complete-Binary Tree

2. Heap property is a binary tree with special characteristics.

It can be classified into two types:

I. Max Heap: If the parent nodes are greater than their child nodes, it is called a Max-Heap.

II. Min Heap: If the parent nodes are smaller than their child nodes, it is called a Min-Heap.

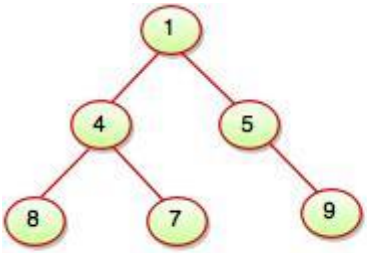


Fig. Min Heap

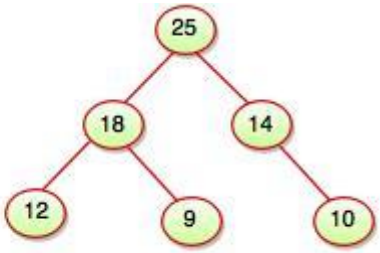


Fig. Max Heap

Algorithm

Step 1 - Construct a **Binary Tree** with given list of Elements.

Step 2 - Transform the Binary Tree into **Min Heap.**

Step 3 - Delete the root element from Min Heap using **Heapify** method.

Step 4 - Put the deleted element into the Sorted list.

Step 5 - Repeat the same until Min Heap becomes empty.

Step 6 - Display the sorted list.

Complexity of the Heap Sort

- Worst Case : O(n log n)
- Best Case : O(n log n)
- Average Case : O(n log n)

Video Content / Details of Itbsite for further learning (if any):
http://www.btechsmartclass.com/data_structures/radix-sort.html
https://nptel.ac.in/courses/106105164/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012, **page nos:** 256-261

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L46**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: Splay Tree

Introduction :

- The topological sorting for a directed acyclic graph is the linear ordering of vertices.
- For every edge U-V of a directed graph, the vertex u will come before vertex v in the ordering.

Prerequisite knowledge for Complete understanding and learning of Topic:

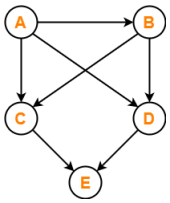
- Sorting
- DAG
- Degree

Detailed content of the Lecture:

- Topological Sort is a linear ordering of the vertices in such a way that if there is an edge in the DAG going from vertex ‘u’ to vertex ‘v’,then ‘u’ comes before ‘v’ in the ordering.

Example

Find the number of different topological orderings possible for the given graph-

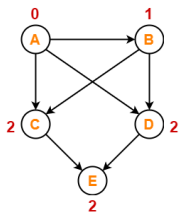


Solution-

The topological orderings of the above graph are found in the following steps-

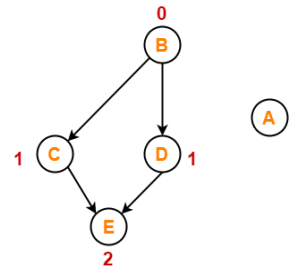
Step-01:

Write in-degree of each vertex-



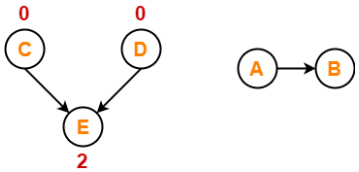
Step-02:

- Vertex-A has the least in-degree.
- So, remove vertex-A and its associated edges.
- Now, update the in-degree of other vertices.



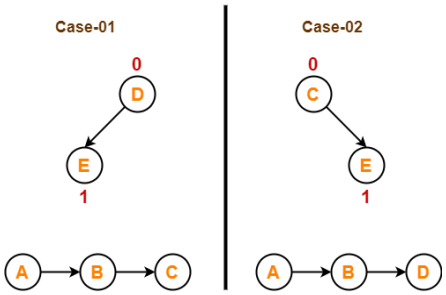
Step-03:

- Vertex-B has the least in-degree.
- So, remove vertex-B and its associated edges.
- Now, update the in-degree of other vertices.



Step-04:

There are two vertices with the least in-degree. So, following 2 cases are possible-



Step-05:

- two cases are continued separately in the similar manner.

CONCLUSION

For the given graph, following **2** different topological orderings are possible

- **ABDCE**
- **ABCDE**

Video Content / Details of website for further learning (if any):

https://www.gatevidyalay.com/topological-sort-topological-sorting/

https://www.youtube.com/watch?v=eL-KzMXSXXI

Important Books/Journals for further learning including the page nos.:

E.Horowitz,S.Sahni Susan , Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008-**page nos:**261-263

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakwa Dist., Tamil Nadu

LECTURE HANDOUTS | **L47**

IT | **II/III**

Course Name with Code : DATA STRUCTURES-19ITC01

Course Teacher : E.Punarselvam

Unit : V **Date of Lecture:**

Topic of Lecture: AVL Tree

Introduction :

- AVL tree is a self-balanced binary search tree. That means, an AVL tree is a binary search tree but it is a balanced tree.
- A binary tree is said to be balanced, if the difference between the heights of left and right subtrees of every node in the tree is either -1, 0 or +1.

Prerequisite knowledge for Complete understanding and learning of Topic:

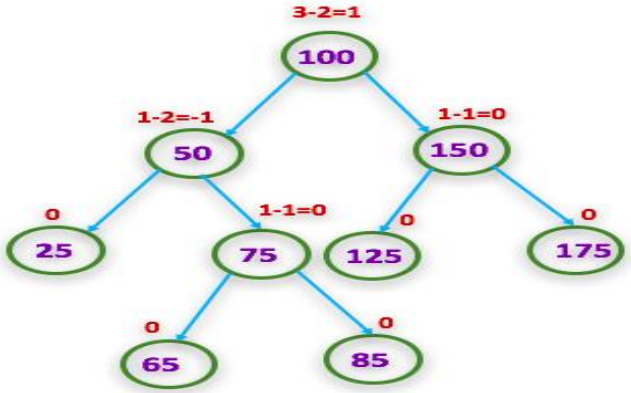
- Binary Tree
- Binary Search Tree
- Complete Binary Tree

Detailed content of the Lecture: Binary tree is said to be balanced if for every node, height of its children differ by at most one. In an AVL tree, every node maintains an extra information known as **balance factor** to take care of the self-balancing nature of the tree.

An AVL tree is defined as follows

- **An AVL tree is a balanced binary search tree. In an AVL tree, balance factor of every node is either -1, 0 or +1**
- **Balance factor = height Of Left Subtree — height Of Right Subtree (OR) height Of Right Subtree — height Of Left Subtree**

An AVL tree is given in the following figure. We can see that, balance factor associated with each node is in between the range of -1 to +1.



AVL Tree

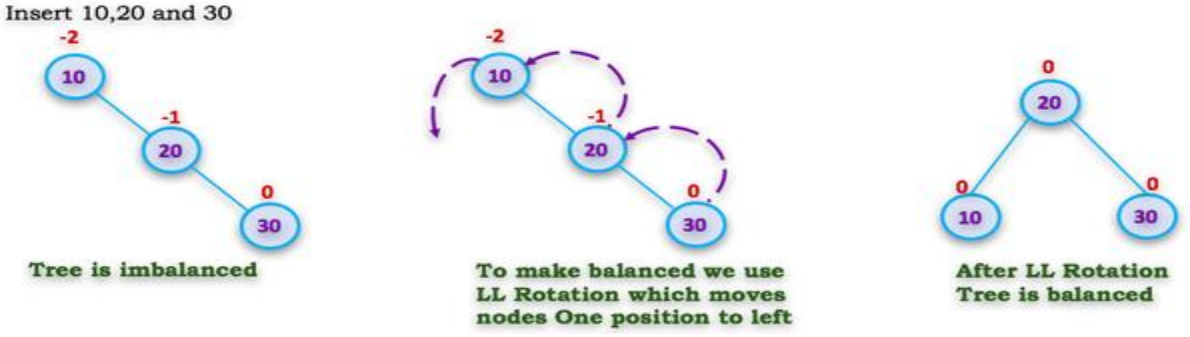
Why AVL Tree ? AVL tree controls the height of the binary search tree by not letting it to be skewed. The time taken for all operations in a binary search tree of height h is O(h). However, it can be extended to O(n) if the BST becomes skewed (i.e. worst case). By limiting this height to log n, AVL tree imposes an upper bound on each operation to be O(log n) where n is the number of nodes.

AVL Tree Rotations ? In AVL tree, after performing every operation like insertion and deletion we need to check the balance factor of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. We use rotation operations to make the tree balanced whenever the tree is becoming imbalanced due to any operation.

Rotation is the process of moving the nodes to either left or right to make tree balanced in terms of its height. To balance itself, an AVL tree may perform the following four kinds of rotations –

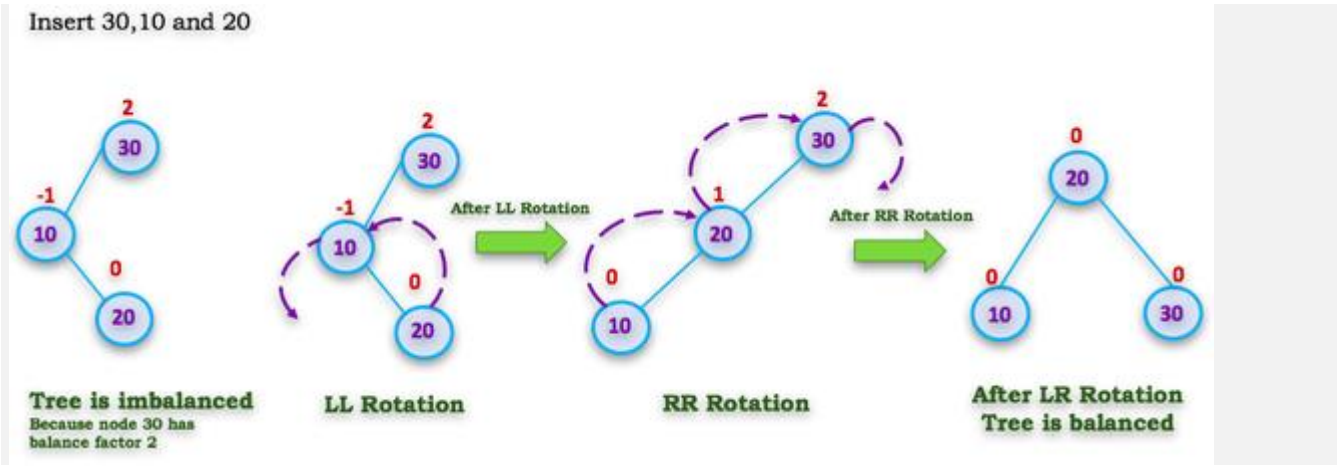
1. Left rotation (Single)
2. Right rotation (Single)
3. Left-Right rotation (Double)
4. Right-Left rotation (Double)

1. Left Rotation (Single LL) In LL Rotation every node moves one position to left from the current position. To understand LL Rotation, let us consider following insertion operations into an AVL Tree…
2.Right Rotation (Single RR) In RR Rotation every node moves one position to right from the current position. To understand RR Rotation, let us consider following insertion operations into an AVL Tree…

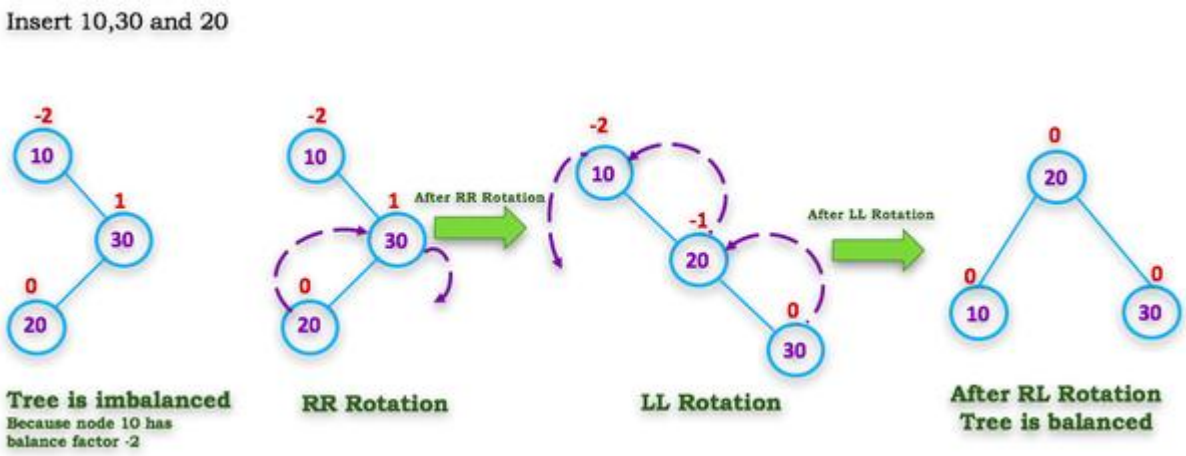


3. Left Right Rotation (Double — LR Rotation) The LR Rotation is combination of single left rotation followed by single right rotation. In LR Rotation, **first every node moves one position to left then one position to right from the current position. To understand LR Rotation, let us consider following insertion operations into an AVL Tree…**

4. Right Left Rotation (Double — RL Rotation)



The RL Rotation is combination of single right rotation followed by single left rotation. In RL Rotation, first every node moves one position to right then one position to left from the current position. To understand RL Rotation, let us consider following insertion operations into an AVL Tree…



Operations on AVL tree Due to the fact that, AVL tree is also a binary search tree therefore, all the operations are performed in the same way as they are performed in a binary search tree. Searching and traversing do not lead to the violation in property of AVL tree. However, insertion and deletion are the operations which can violate this property.

1. Searching Searching in an AVL Tree is done as in any binary search tree. The Special thing about AVL Tree is that the number of comparison required, i.e. The AVL three's height, is guaranteed never to exceed log(n).

Step 1: Read the search element from the user.

Step 2: Compare, the search element with the value of root node in the tree.

Step 3: If both are matching, then display "Given node found!!!" and terminate the function.

Step 4: If both are not matching, then check whether search element is smaller or larger than that node value.

Step 5: If search element is smaller, then continue the search process in left subtree.

Step 6: If search element is larger, then continue the search process in right subtree.

Step 7: Repeat the same until we found exact element or we completed with a leaf node.

Step 8: If we reach to the node with search value, then display "Element is found" and terminate the function.

Step 9: If we reach to a leaf node and it is also not matching, then display "Element not found" and terminate the function.

2. Insertion

Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.

Step 1: Insert the new element into the tree using Binary Search Tree insertion logic.
Step 2: After insertion, check the Balance Factor of every node.
Step 3: If the Balance Factor of every node is 0 or 1 or -1 then go for next operation.
Step 4: If the Balance Factor of any node is other than 0 or 1 or -1 then tree is said to be imbalanced. Then perform the suitable Rotation to make it balanced. And go for next operation.

3. Deletion

Deletion can also be performed in the same way as it is performed in a binary search tree. Deletion may also disturb the balance of the tree therefore; various types of rotations are used to rebalance the tree.

Important Books/Journals for further learning including the page nos.:

E.Horowitz,S.Sahni Susan , Anderson-Freed, Fundamentals of Data structures in C, Universities Press.2008- **page nos:**261-263

Course Teacher

Verified by HOD