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

LECTURE HANDOUTS

L 01

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 04.08.2020

Topic of Lecture: Introduction

Introduction : (Maximum 5 sentences)

- **Data Structure** can be defined as the group of **data** elements which provides an efficient way of storing and organizing **data** in the computer so that it can be used efficiently.
- Some examples of **Data Structures** are arrays, Linked List, Stack, Queue, etc.
- **Data Structure** is a way of organizing all **data** items that considers not only the elements stored but also. their relationship to each other.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of mathematics
- knowledge of programming languages
- Data types

Detailed content of the Lecture:

- Data Structure is a systematic way to organize data in order to use it efficiently. Following terms are the foundation terms of a data structure.
- **Interface** − Each data structure has an interface. Interface represents the set of operations that a data structure supports.
- An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.
- **Implementation** − Implementation provides the internal representation of a data structure. Implementation also provides the definition of the algorithms used in the operations of the data structure.

Characteristics of a Data Structure

- **Correctness** − Data structure implementation should implement its interface correctly.
- **Time Complexity** − Running time or the execution time of operations of data structure must be as small as possible.

- **Space Complexity** − Memory usage of a data structure operation should be as little as possible.

Why to Learn Data Structure and Algorithms

As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

- **Data Search** − Consider an inventory of 1 million(10^6) items of a store. If the application is to search an item, it has to search an item in 1 million(10^6) items every time slowing down the search. As data grows, search will become slower.
- **Processor speed** − Processor speed although being very high, falls limited if the data grows to billion records.
- **Multiple requests** − As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

To solve the above-mentioned problems, data structures come to rescue. Data can be organized in a data structure in such a way that all items may not be required to be searched, and the required data can be searched almost instantly.

Applications of Data Structure and Algorithms

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.
- Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms −

- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.

The following computer problems can be solved using Data Structures −

- Fibonacci number series
- Knapsack problem
- Tower of Hanoi
- All pair shortest path by Floyd-Warshall

- Shortest path by Dijkstra
- Project scheduling

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

- https://www.geeksforgeeks.org/data-structures/
- https://www.youtube.com/watch?v=xLetJpcjHS0

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

Alfred V.Aho, John E.Hopcroft,Jeffrey D.Ullman , ,"Data Structures and Algorithms" Pearson Education" – Page No -15

Course Faculty

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

LECTURE HANDOUTS

L 02

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 05.08.2020

Topic of Lecture: Abstract Data Types (ADT)

Introduction : (Maximum 5 sentences)

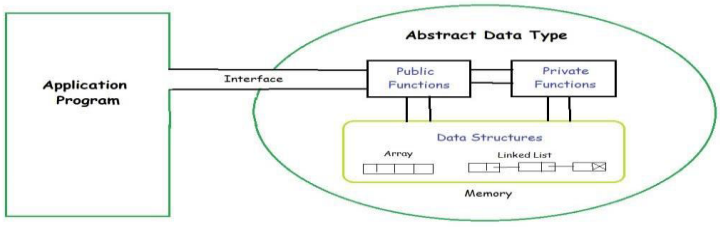
- **Abstract Data type (ADT)** is a type (or class) for objects whose behaviour is defined by a **set of value and a set of operations.**
- The definition of ADT only mentions **what operations are to be performed but not how these operations will be implemented.**

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of mathematics
- knowledge of programming languages
- Data types

Detailed content of the Lecture:

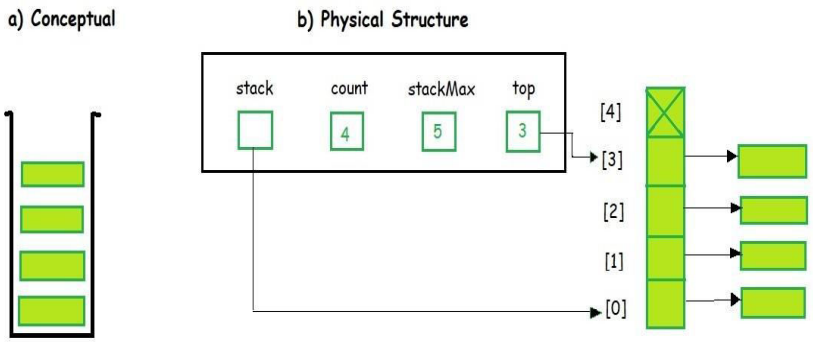
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
- **ADT Diagram**



- Three ADTs namely List ADT, Stack ADT, Queue ADT

Stack ADT

- In Stack ADT Implementation instead of data being stored in each node, the pointer to data is stored.
- The program allocates memory for the *data* and *address* is passed to the stack ADT.



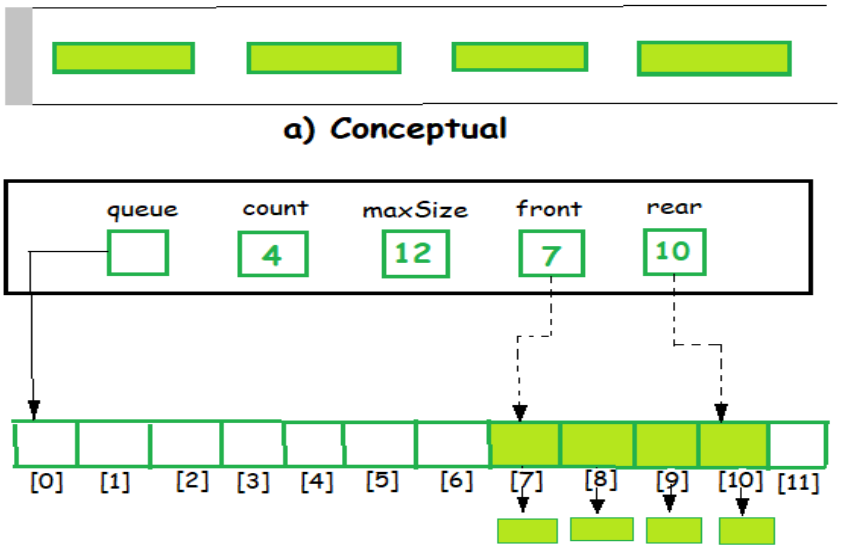
- The head node and the data nodes are encapsulated in the ADT. The calling function can only see the pointer to the stack.
- The stack head structure also contains a pointer to *top* and *count* of number of entries currently in stack.

//Stack ADT Type Definitions

```
typedef struct node
{  void *DataPtr;
 struct node *link;
} StackNode;
typedef struct
{  int count;
 StackNode *top;
} STACK;
```

Queue ADT

- The queue abstract data type (ADT) follows the basic design of the stack abstract data type.
- Each node contains a void pointer to the *data* and the *link pointer* to the next element in the queue. The program's responsibility is to allocate memory for storing the data.



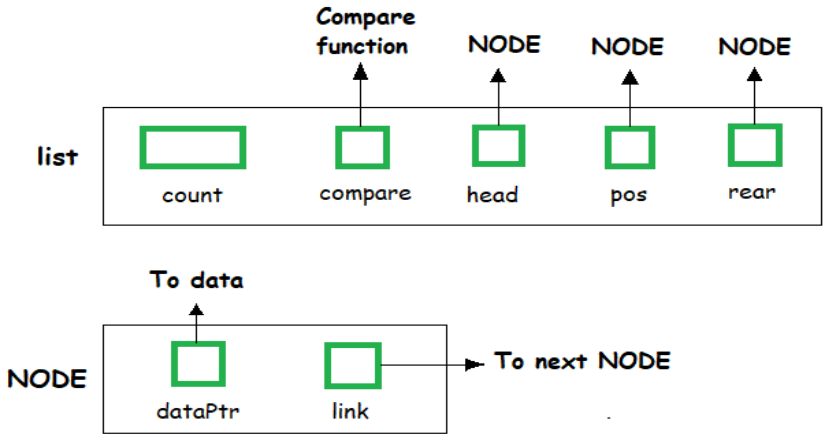
//Queue ADT Type Definitions

```
typedef struct node
{  void *DataPtr;
 struct node *next;
} QueueNode;
typedef struct
{
 QueueNode *front;
 QueueNode *rear;
 int count;
} QUEUE;
```

Linked List ADT

The data is generally stored in key sequence in a list which has a head structure consisting of *count*, *pointers* and *address of compare function* needed to compare the data in the list.

- The data node contains the *pointer* to a data structure and a *self-referential pointer* which points to the next node in the list.



//List ADT Type Definitions

```
typedef struct node
{  void *DataPtr;
 struct node *link;
} Node;
typedef struct
{
 int count;
 Node *pos;
 Node *head;
 Node *rear;
 int (*compare) (void *argument1, void *argument2)
} LIST;
```

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

- https://www.tutorialspoint.com/abstract-data-type-in-data-structures

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

- Alfred V.Aho, John E.Hopcroft,Jeffrey D.Ullman , ,"Data Structures and Algorithms" Pearson Education" – Page No -24

Course Faculty

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

LECTURE HANDOUTS

L 03

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 06.08.2020

Topic of Lecture: Arrays and its representation

Introduction : (Maximum 5 sentences)

- An **Array** is a Linear data structure which is a collection of data items having similar data types stored in contiguous memory locations.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Defining An Array
- Types of Data

Detailed content of the Lecture:

Array Operations

- Traverse − Print all the elements in the array one by one.
- Insertion − Adds an element at the given index.
- Deletion − Deletes an element at the given index.
- Search − Searches an element in the array using the given index or the value.
- Update − Updates an element at the given index.

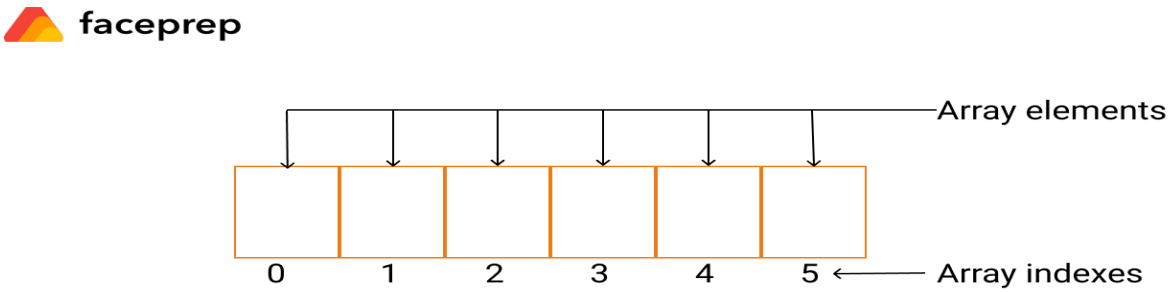
Types of Arrays

- The various types of arrays are as follows.
- One dimensional array
- Multi-dimensional array

One-Dimensional Array

- A one-dimensional array is also called a single dimensional array where the elements will be accessed in sequential order.
- This type of array will be accessed by the subscript of either a column or row index.
- **Arrays and its representation** is given below **Array** Index: The location of an element in

an **array** has an index, which identifies the element. **Array** index starts from 0.



Array Length: The length of an array is defined based on the number of elements an array can store. In the above example, array length is 6 which means that it can store 6 elements.

int a[6];

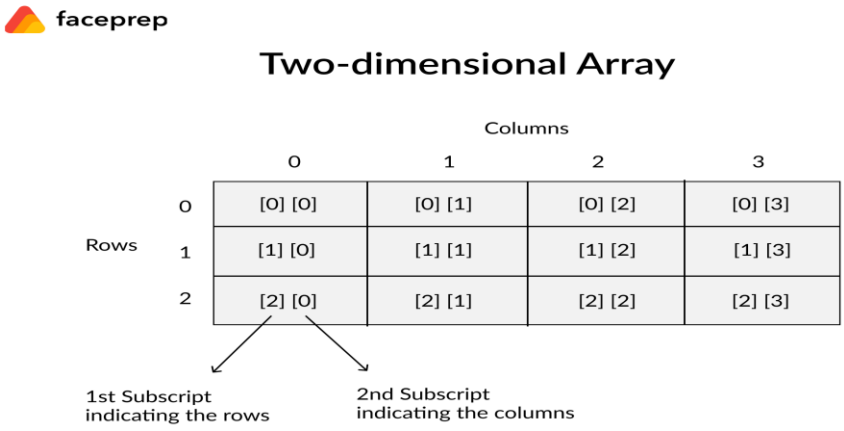
Int as Data type

Total no of Elements =6

Array starts from 0 to 5

Multi-Dimensional Array

- When the number of dimensions specified is more than one, then it is called as a multi-dimensional array. Multidimensional arrays include 2D arrays and 3D arrays.
- Two-dimensional array **face [3] [4],** the first index specifies the number of rows and the second index specifies the number of columns and the array can hold 12 elements (3 * 4).
-



-

Declaration/Initialization of Arrays

- // A sample program for Array Declaration

```
#include <stdio.h>

int main()

{

int one_dim [10];

int  two_dim [2][2];

int  three_dim [2][3][4] =

{

{ {3, 4, 2, 3},   {0, -3, 9, 11},  {23, 12, 23, 2} },

 { {13, 4, 56, 3},  {5, 9, 3, 5},  {3, 1, 4, 9} }

    };

#declaration of 3D array. Here the elements are also defined. return 0;

}
```

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

- https://www.geeksforgeeks.org/introduction-to-arrays/

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

- Alfred V.Aho, John E.Hopcroft,Jeffrey D.Ullman , ,"Data Structures and Algorithms" Pearson Education" – Page No -27

Course Faculty

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

LECTURE HANDOUTS

L 04

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 07.08.2020

Topic of Lecture: Structures

Introduction : (Maximum 5 sentences)

- Data Structure can be defined as the group of data elements which provides an efficient way of storing and organising data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of mathematics
- knowledge of programming languages
- Data types

Detailed content of the Lecture:

- Data Structure is a way to store and organize data so that it can be used efficiently.
- Our Data Structure tutorial includes all topics of Data Structure such as Array, Pointer, Structure, Linked List, Stack, Queue, Graph, Searching, Sorting, Programs, etc.
- The data structure name indicates itself that organizing the data in memory.
- There are many ways of organizing the data in the memory as we have already seen one of the data structures, i.e., array in C language.
- Array is a collection of memory elements in which data is stored sequentially, i.e., one after another.
- In other words, we can say that array stores the elements in a continuous manner.
- This organization of data is done with the help of an array of data structures.
- There are also other ways to organize the data in memory.
- The data structure is not any programming language like C, C++, java, etc.
- It is a set of algorithms that we can use in any programming language to structure the data in the memory.
- To structure the data in memory, 'n' number of algorithms were proposed, and all these algorithms are known as Abstract data types.
- These abstract data types are the set of rules.

Types of Data Structures

There are two types of data structures:

- Primitive data structure

- Non-primitive data structure

Primitive Data structure

- The primitive data structures are primitive data types. The int, char, float, double, and pointer are the primitive data structures that can hold a single value.

Non-Primitive Data structure

- The non-primitive data structure is divided into two types:
- Linear data structure
- Non-linear data structure

Linear Data Structure

- The arrangement of data in a sequential manner is known as a linear data structure. The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues. In these data structures, one element is connected to only one another element in a linear form.

Data structures can also be classified as:

- **Static data structure:** It is a type of data structure where the size is allocated at the compile time. Therefore, the maximum size is fixed.
- **Dynamic data structure:** It is a type of data structure where the size is allocated at the run time. Therefore, the maximum size is flexible.
- Major Operations
- The major or the common operations that can be performed on the data structures are:
- **Searching:** We can search for any element in a data structure.
- **Sorting:** We can sort the elements of a data structure either in an ascending or descending order.
- **Insertion:** We can also insert the new element in a data structure.
- **Updation:** We can also update the element, i.e., we can replace the element with another element.
- **Deletion:** We can also perform the delete operation to remove the element from the data structure.

Advantages of Data structures

The following are the advantages of a data structure:

- **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
- **Reusability:** The data structure provides reusability means that multiple client programs can use the data structure.

- **Abstraction:** The data structure specified by an ADT also provides the level of abstraction.
- The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

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

- https://www.javatpoint.com/data-structure-tutorial

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -3 Page No -3.2

Course Faculty

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

LECTURE HANDOUTS

L 05

MCA

I/I

Course Name with Code : **19CAB02 & Data Structures and Algorithms**

Course Faculty : **G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 10.08.2020

Topic of Lecture: Stack Infix to postfix conversion – evaluation of expression

Introduction : (Maximum 5 sentences)

- Stack is **a linear data structure which follows a particular order in which the operations are performed**. The order may be LIFO(Last In First Out) or FILO(First In Last Out). ... So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Structure
- Linear Data Structure

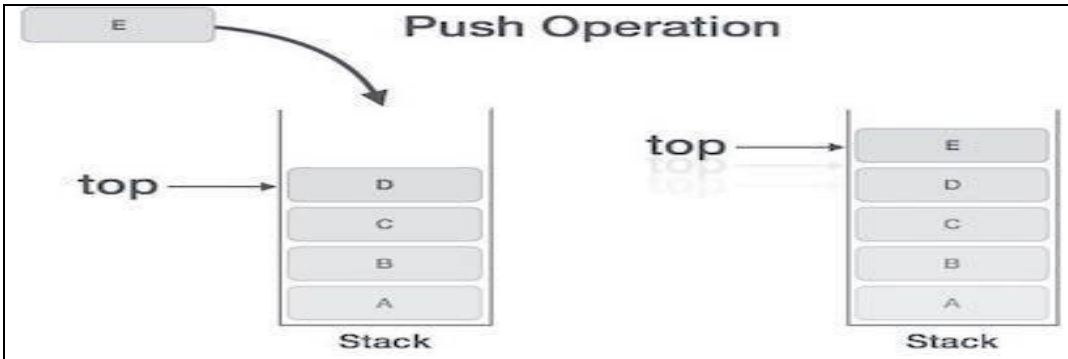
Detailed content of the Lecture:

Stack Operations

- Stack operations may involve initializing the stack, using it and then de-initializing it.
- Apart from these basic stuffs, a stack is used for the following two primary operations −
- push() − Pushing (storing) an element on the stack.
- pop() − Removing (accessing) an element from the stack.
- When data is PUSHed onto stack.
- To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −
- peek() − get the top data element of the stack, without removing it.
- isFull() − check if stack is full.
- isEmpty() − check if stack is empty.

Push() operation

- The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps −
- Step 1 − Checks if the stack is full.
- Step 2 − If the stack is full, produces an error and exit.
- Step 3 − If the stack is not full, increments top to point next empty space.
- Step 4 − Adds data element to the stack location, where top is pointing.
- Step 5 − Returns success.
- If the linked list is used to implement the stack, then in step 3, we need to allocate space dynamically.
-



- If the linked list is used to implement the stack, then in step 3, we need to allocate space dynamically.

Push operation algorithm

begin procedure push: stack, data
if stack is full
return null
End if
top ← top + 1
stack[top] ← data
end procedure

Pop() operation

- Accessing the content while removing it from the stack, is known as a Pop Operation.
- In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value.
- But in linked-list implementation, pop() actually removes data element and deallocates memory space.

Video Content / Details of website for further learning (if any):
https://onlinecourses.nptel.ac.in/noc19_cs64/unit?unit=6&lesson=11

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -5 Page No -5.27

Course Faculty

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

LECTURE HANDOUTS

L 06

MCA

III/V

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 11.08.2020

Topic of Lecture: Applications of stack

Introduction : (Maximum 5 sentences)

- We can implement a stack and queue using both array and linked list. Stack Applications: **During Function Calls and Recursive Algorithms, Expression Evaluation, Undo feature in computer keyboard**, Converting an Infix to Postfix, During Depth First Search (DFS) and Backtracking Algorithms etc.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Types of Structure
- Stack

Detailed content of the Lecture:

Following are some of the important applications of a Stack data structure:

1. Stacks can be used for expression evaluation.
2. Stacks can be used to check parenthesis matching in an expression.
3. Stacks can be used for Conversion from one form of expression to another.
4. Stacks can be used for Memory Management.
5. Stack data structures are used in backtracking problems.

Expression Evaluation

Stack data structure is used for evaluating the given expression. For example, consider the following expression

5 * (6 + 2) - 12 / 4

Since parenthesis has the highest precedence among the arithmetic operators,

(6 +2) = 8 will be evaluated first. Now, the expression becomes

5 * 8 - 12 / 4

* and / have equal precedence and their associativity is from left-to-right.

So, start evaluating the expression from left-to-right.

5 * 8 = 40 and **12 / 4 = 3**

Now, the expression becomes

40 - 3

And the value returned after the subtraction operation is **37**.

Parenthesis Matching

Given an expression, you have to find if the parenthesis is either correctly matched or not. For example, consider the expression **(a + b) * (c + d).**

Expression Conversion

Converting one form of expressions to another is one of the important applications of stacks.

Memory management

The assignment of memory takes place in contiguous memory blocks. We call this stack memory allocation because the assignment takes place in the function call stack. The size of the memory to be allocated is known to the compiler. When a function is called, its variables get memory allocated on the stack.

Backtracking Problems

Consider the **N-Queens problem** for an example. The solution of this problem is that N queens should be positioned on a chessboard so that none of the queens can attack another queen.

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

- https://www.faceprep.in/data-structures/stack-applications-in-data-structure/

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -5 Page No -5.19

Course Faculty

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

LECTURE HANDOUTS

L 07

MCA

III/V

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 12.08.2020

Topic of Lecture: Queue , Circular Queue , Applications of Queue

Introduction : (Maximum 5 sentences)

- 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.
- Queue follows the FIFO (First - In - First Out) structure.

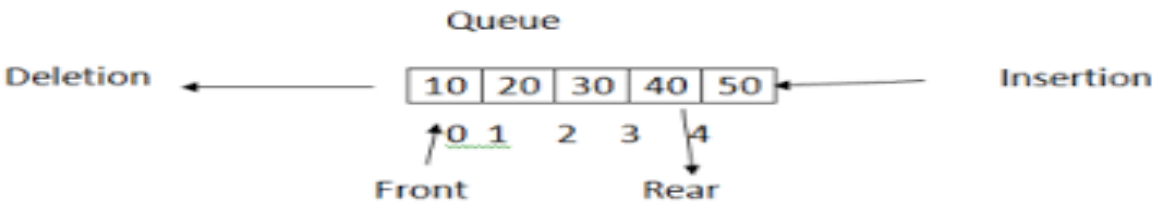
Prerequisite knowledge for Complete understanding and learning of Topic:

- Types of Linear Data structure

Detailed content of the Lecture:

- Queue follows the FIFO (First - In - First Out) structure.
- According to its FIFO structure, element inserted first will also be removed first.

Diagram for Queue



- In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue), because queue is open at both its ends.
- The enqueue() and dequeue() are two important functions used in a queue.

Operations on Queue

- **Following are the basic operations performed on a Queue.**

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.

Queue Implementation

- In the Below diagram, Front and Rear of the queue point at the first index of the array. (Array index starts from 0).
- While adding an element into the queue, the Rear keeps on moving ahead and always points to the position where the next element will be inserted. Front remains at the first index.

Queue Example

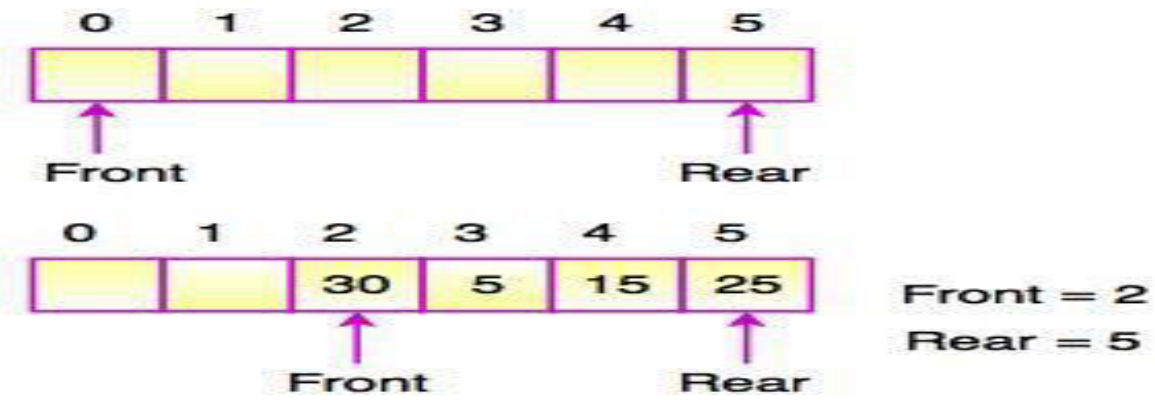


Fig. Implementation of Queue using Array

Simple Queue

- Simple queue defines the simple operation of queue in which **insertion** occurs at the **rear** of the

list and **deletion** occurs at the **front** of the list.

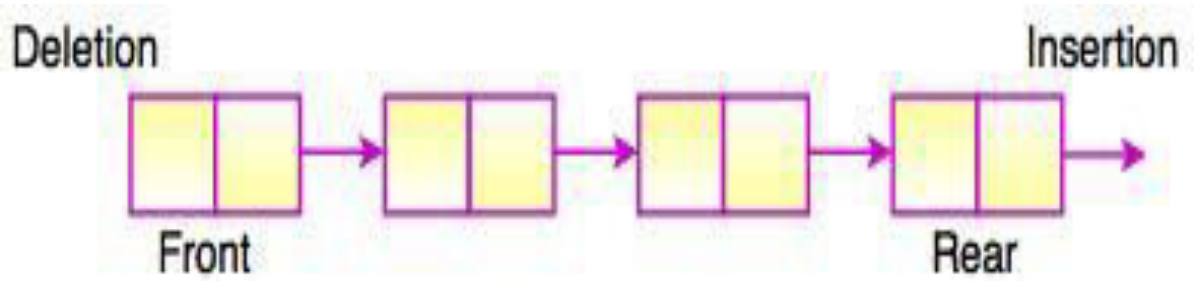


Fig. Simple Queue

Circular Queue

- In a circular queue, all nodes are treated as circular. Last node is connected back to the first node.
- Circular queue is also called as Ring Buffer.
- It is an abstract data type.
- Circular queue contains a collection of data which allows insertion of data at the end of the queue and deletion of data at the beginning of the queue.

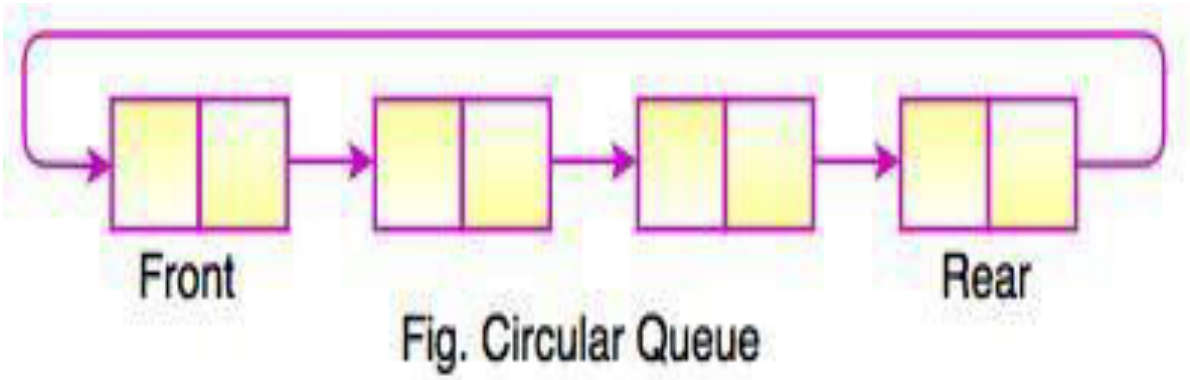


Fig. Circular Queue

- The above figure shows the structure of circular queue. It stores an element in a circular way and performs the operations according to its FIFO structure

Deque (Double Ended Queue)

- In Double Ended Queue, insert and delete operation can be occur at both ends that is front and rear of the queue.

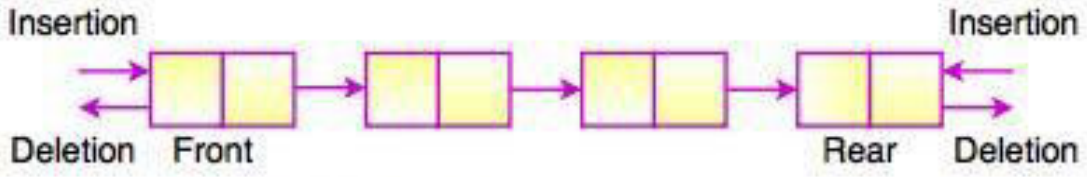
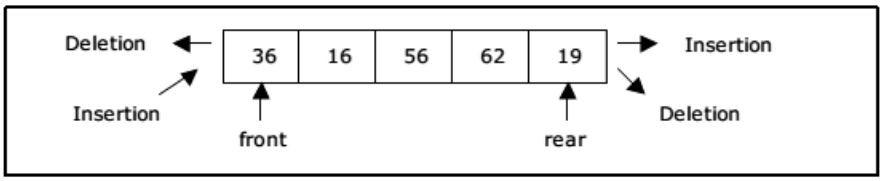


Fig. Double Ended Queue (Dequeue)

Data Structure – Deque Overview

In the preceding section we saw that a queue in which we insert items at one end and from which we remove items at the other end. In this section we examine an extension of the queue,Deque ,which provides a means to insert and remove items at both ends of the queue. This data structure is a deque. The word deque is an acronym derived from double-ended queue. Figure 4.5 shows the representation of a deque.



A deque provides four operations. Figure 4.6 shows the basic operations on a deque.

- **enqueue_front**: insert an element at front.
- **dequeue_front**: delete an element at front.
- **enqueue_rear**: insert element at rear.
- **dequeue_rear**: delete element at rear.

Priority Queue.

- Priority queue contains data items which have some preset priority. While removing an element from a priority queue, the data item with the highest priority is removed first.
- In a priority queue, insertion is performed in the order of arrival and deletion is performed based on the priority.

Queue Advantages

- It is able to handle multiple data types and they are both flexible and flexibility and fast. Moreover, **queues** can be of potentially infinite length compared with the use of fixed-length arrays.

Disadvantages

- The **problem** that arises with the **linear queue** is that if some empty cells occur at the beginning of the **queue** then we cannot insert new element at the empty space as the rear cannot be further incremented.
- **Applications of Queue**

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

- **https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm**
- **https://www.geeksforgeeks.org/queue-data-structure/**

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -5 Page No -5.42

Course Faculty

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 08

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 13.08.2020

Topic of Lecture: Linked Lists , Doubly Linked lists

Introduction : (Maximum 5 sentences)

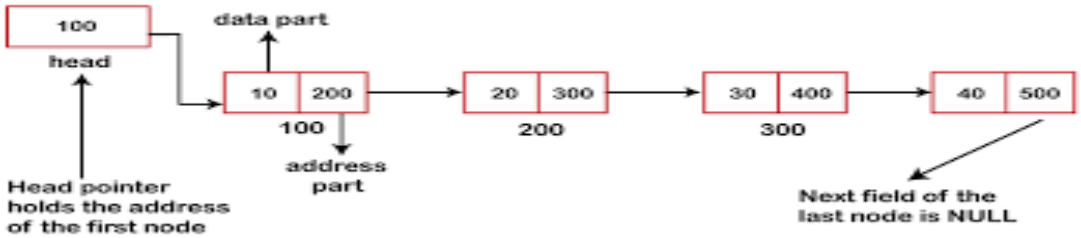
- A linked list is a sequence of data structures, which are connected together via links.
- Linked List is a sequence of links which contains items.
- Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Prerequisite knowledge for Complete understanding and learning of Topic:

- **Basics of Data types**
- **Introduction of Linear Data structure**

Detailed content of the Lecture:

- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.



Uses of Linked List

- The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
- list size is limited to the memory size and doesn't need to be declared in advance.

- Empty node can not be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Singly linked list or One way chain

- Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program.
- A node in the singly linked list consist of two parts: data part and link part.
- Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.
- One way chain or singly linked list can be traversed only in one direction.
- In other words, we can say that each node contains only next pointer, therefore we can not traverse the list in the reverse direction.
- A linked list is a sequence of data structures, which are connected together via links.
- Linked List is a sequence of links which contains items.
- Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.
- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **LinkedList** − A Linked List contains the connection link to the first link called First.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.

As per the above illustration, following are the important points to be considered.

- Linked List contains a link element called first.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Types of Linked List

Following are the various types of linked list.

- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

Following are the basic operations supported by a list.

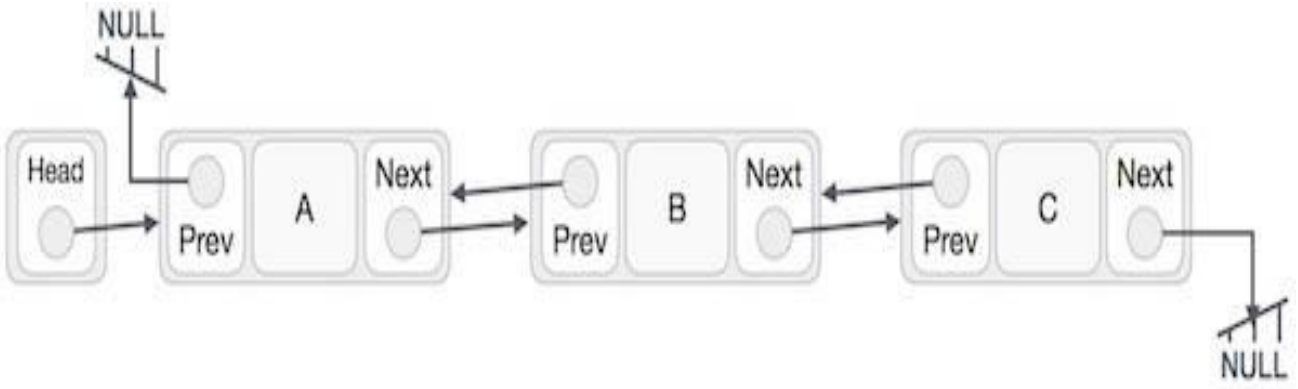
- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Display** − Displays the complete list.
- **Search** − Searches an element using the given key.
- **Delete** − Deletes an element using the given key.

DOUBLY LINKED LIST

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **Prev** − Each link of a linked list contains a link to the previous link called Prev.
- **LinkedList** − A Linked List contains the connection link to the first link called First and to the last link called Last.
-

Doubly Linked List Representation



As per the above illustration, following are the important points to be considered.

- Doubly Linked List contains a link element called first and last.
- Each link carries a data field(s) and two link fields called next and prev.
- Each link is linked with its next link using its next link.
- Each link is linked with its previous link using its previous link.
- The last link carries a link as null to mark the end of the list.

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

- **https://www.geeksforgeeks.org/data-structures/linked-list**
- **https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm**
- **https://www.javatpoint.com/singly-linked-list**

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -4 Page No - 4.2

Course Faculty

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 09

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 14.08.2020

Topic of Lecture: Applications of linked list , Polynomial Addition.

Introduction : (Maximum 5 sentences)

- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Introduction of Data structure
- Basic of Linear Structure

Detailed content of the Lecture:

- Useful for implementation of queue.
- Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list.
- We can maintain a pointer to the last inserted node and front can always be obtained as next of last.
- Circular lists are useful in applications to repeatedly go around the list.
- For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application.
- It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.
- Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

POLYNOMIAL ADDITION USING LINKED LIST

Adding two polynomials using Linked List

Given two polynomial numbers represented by a linked list. Write a function that add these lists means add the coefficients who have same variable powers.

Example:

Input:

1st number = $5x^2 + 4x^1 + 2x^0$
2nd number = $-5x^1 - 5x^0$

Output:

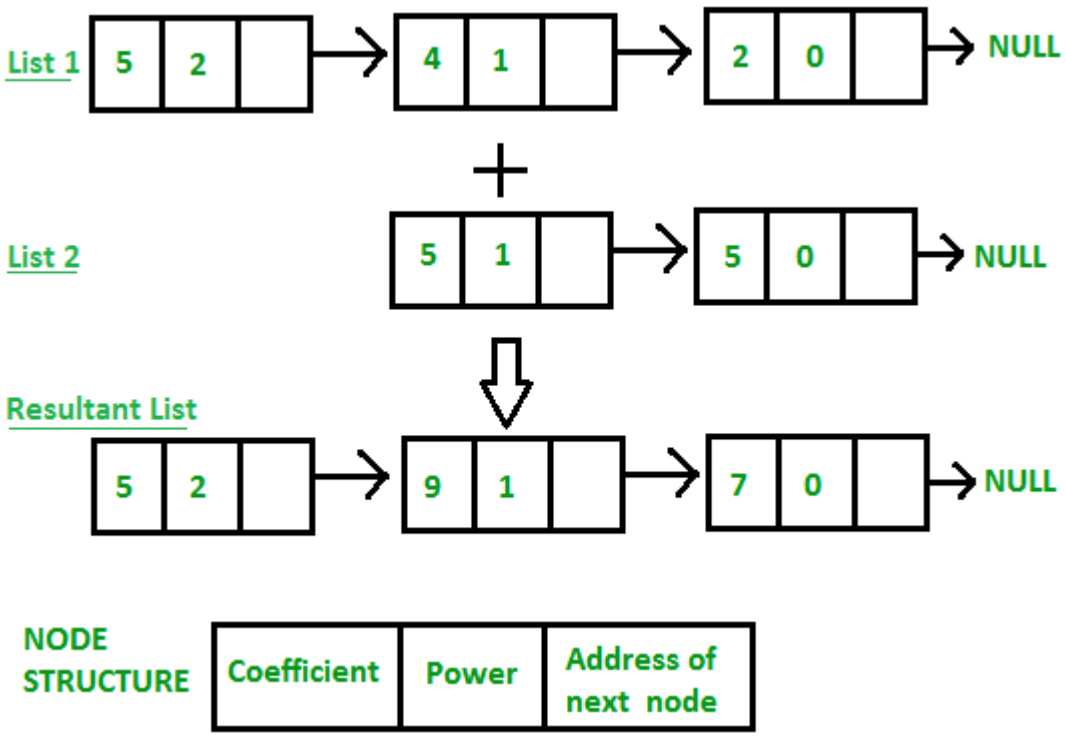
$5x^2-1x^1-3x^0$

Input:

1st number = $5x^3 + 4x^2 + 2x^0$
2nd number = 5x^1 - 5x^0

Output:

$5x^3 + 4x^2 + 5x^1 - 3x^0$



Output

```
1st Number: 5x^2+4x^1+2x^0
2nd Number: -5x^1-5x^0
Added polynomial: 5x^2-1x^1-3x^0
```

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

- https://www.javatpoint.com/doubly-linked-list
- https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm
- https://www.programiz.com/dsa/linked-list-types#doubly

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -4 Page No -4.118
-

Course Faculty

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

LECTURE HANDOUTS

L 10

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **I - Linear Data Structures**

Date of Lecture: 16.08.2020

Topic of Lecture: Need for non-linear structures

Introduction :

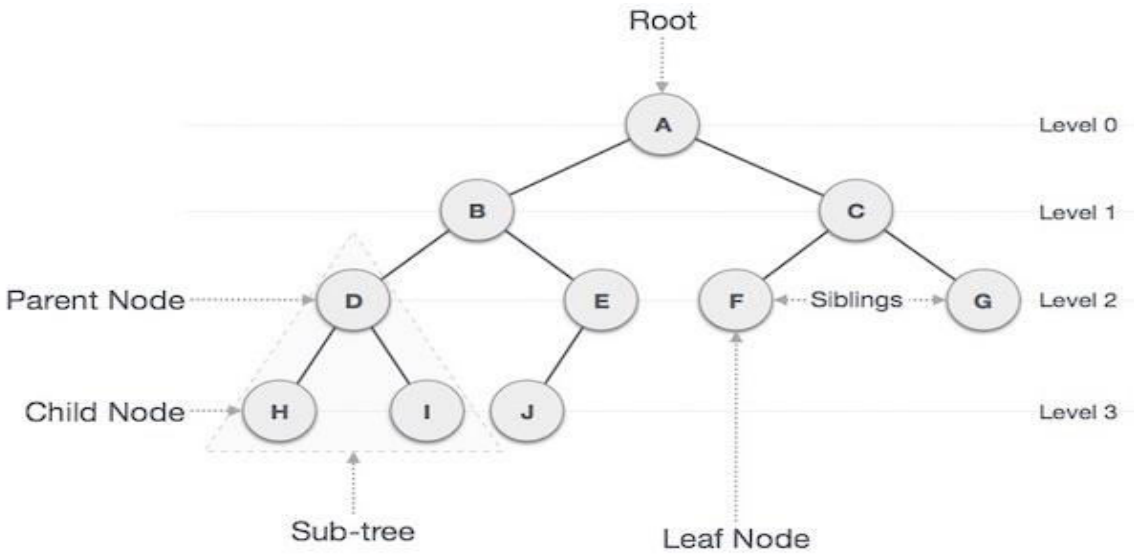
- A data structure is said to be non linear if its elements form a hierarchical classification where, data items appear at various levels.
- Trees and Graphs are widely used non-linear data structures.
- Tree and graph structures represent hierarchical relationship between individual data elements

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of structure
- Types of Non-Linear Data structures

Detailed content of the Lecture:

- Trees represent a special case of more general structures known as graphs.
- In a graph, there is no restrictions on the number of links that can enter or leave a node, and cycles may be present in the graph.



- Tree is a popular data structure used in wide range of applications.
- A tree data structure can be defined as follows...
- Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.
- A tree data structure can also be defined as follows...
- A tree is a finite set of one or more nodes such that:

- There is a specially designated node called the root.
- The remaining nodes are partitioned into n>=0 disjoint sets T1, ..., Tn, where each of these sets is a tree. We call T1, ..., Tn are the subtrees of the root.
- A tree is hierarchical collection of nodes.
- One of the nodes, known as the root, is at the top of the hierarchy.
- Each node can have at most one link coming into it.
- The node where the link originates is called the parent node. The root node has no parent.
- The links leaving a node (any number of links are allowed) point to child nodes.
- Trees are recursive structures. Each child node is itself the root of a subtree.
- At the bottom of the tree are leaf nodes, which have no children.

Advantages of trees

- Trees are so useful and frequently used, because they have some very serious advantages:
- Trees reflect structural relationships in the data
- Trees are used to represent hierarchies
- Trees provide an efficient insertion and searching
- Trees are very flexible data, allowing to move sub trees around with minimum effort

1.Root

- In a tree data structure, the first node is called as Root Node.
- Every tree must have root node.
- We can say that root node is the origin of tree data structure.
- In any tree, there must be only one root node.
- We never have multiple root nodes in a tree. In above tree, A is a Root node

2. Edge

- In a tree data structure, the connecting link between any two nodes is called as EDGE.
- In a tree with 'N' number of nodes there will be a maximum of 'N-1' number of edges.

3. Parent

- In a tree data structure, the node which is predecessor of any node is called as PARENT NODE.
- In simple words, the node which has branch from it to any other node is called as parent node.
- Parent node can also be defined as "The node which has child / children". e.g., Parent (A,B,C,D).

4. Child

- In a tree data structure, the node which is descendant of any node is called as CHILD Node.
- In simple words, the node which has a link from its parent node is called as child node.
- In a tree, any parent node can have any number of child nodes.
- In a tree, all the nodes except root are child nodes. e.g., Children of D are (H, I,J).

5. Siblings

- In a tree data structure, nodes which belong to same Parent are called as SIBLINGS.
- In simple words, the nodes with same parent are called as Sibling nodes.
- Ex: Siblings (B,C, D)**.**

6. Leaf

- In a tree data structure, the node which does not have a child (or) node with degree zero is called as LEAF Node.
- In simple words, a leaf is a node with no child.

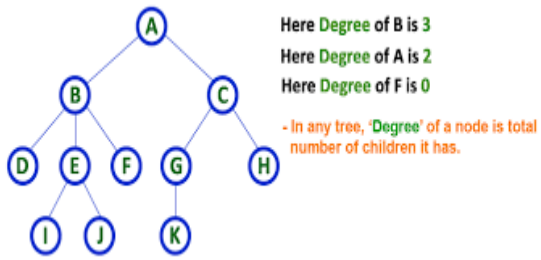
7. Internal Nodes

- In a tree data structure, the node which has atleast one child is called as INTERNAL Node.
- In simple words, an internal node is a node with atleast one child.
- In a tree data structure, nodes other than leaf nodes are called as Internal Nodes.
- The root node is also said to be Internal Node if the tree has more than one node.
- Internal nodes are also called as 'Non-Terminal' nodes. Ex:B,C,D,E,H.

8. Degree

- In a tree data structure, the total number of children of a node (or)number of subtrees of a node is called as DEGREE of that Node.
- In simple words, the Degree of a node is total number of children it has.

- The highest degree of a node among all the nodes in a tree is called as 'Degree of Tree'
-



9. Level

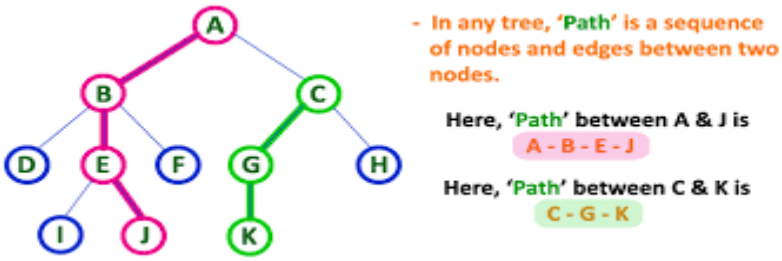
- In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on...
- In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step). Some authors start root level with 1.

10. Height

- In a tree data structure, the total number of edges from leaf node to a particular node in the longest path is called as HEIGHT of that Node.
- In a tree, height of the root node is said to be height of the tree.
- In a tree, height of all leaf nodes is '0'. 11.
- Depth In a tree data structure, the total number of edges from root node to a particular node is called as DEPTH of that Node.
- In a tree, the total number of edges from root node to a leaf node in the longest path is said to be Depth of the tree.
- In simple words, the highest depth of any leaf node in a tree is said to be depth of that tree. In a tree, depth of the root node is '0'.

12. Path

- In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as PATH between that two Nodes.



- Length of a Path is total number of nodes in that path. In below example the path A - B - E - J has length 4.

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

1. https://tildesites.bowdoin.edu/~ltoma/teaching/cs210/spring09/Slides/210-Trees.pdf
2. https://www.cs.purdue.edu/homes/ayg/CS251/slides/chap5.pdf
3. https://www.pvpsiddhartha.ac.in/dep_it/lecture%20notes/CDS/unit4.pdf

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -5 Page No -6.2

Course Faculty

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

LECTURE HANDOUTS

L 11

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **II - Tree Structures**

Date of Lecture: 24.08.2020

Topic of Lecture: Trees and its representation

Introduction : (Maximum 5 sentences)

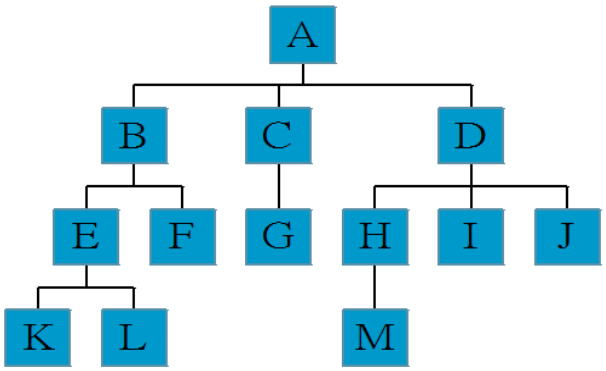
- A tree data structure can be represented in two methods. Those methods are as follows...
- 1.List Representation
- 2. Left Child - Right Sibling Representation Consid

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Trees

Detailed content of the Lecture:

- A tree data structure can be represented in two methods.



1. List Representation

- In this representation, we use two types of nodes one for representing the node with data and another for representing only references.
- We start with a node with data from root node in the tree.
- Then it is linked to an internal node through a reference node and is linked to any other node directly.
- This process repeats for all the nodes in the tree.
- The above tree example can be represented using List representation as follows...

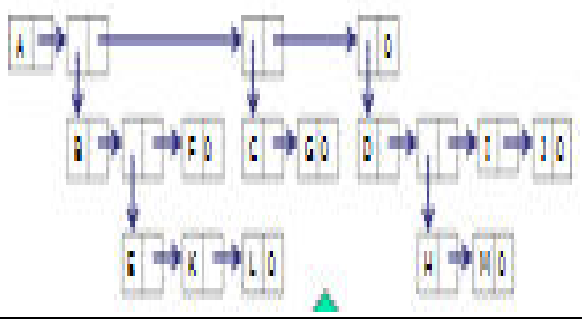


Fig: List representation of above Tree

List Representation

– (A (B (E (K, L), F), C (G), D (H (M), I, J)))

– The root comes first, followed by a list of sub-trees

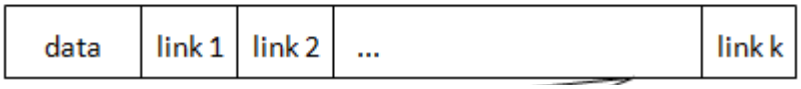


Fig: Possible node structure for a tree of degree k

2. Left Child - Right Sibling Representation

- In this representation, we use list with one type of node which consists of three fields namely Data field, Left child reference field and Right sibling reference field.
- Data field stores the actual value of a node, left reference field stores the address of the left child and right reference field stores the address of the right sibling node.

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

1. **https://www.pvpsiddhartha.ac.in/dep_it/lecture%20notes/CDS/unit4.pdf**

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -6 Page No - 6.5

Course Faculty

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

LECTURE HANDOUTS

L 12

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **II - Tree Structures**

Date of Lecture: 25.08.2020

Topic of Lecture: Binary Tree

Introduction : (Maximum 5 sentences)

- Binary tree is a special type of tree data structure in which every node can have a maximum of 2 children.
- One is known as left child and the other is known as right child.
- A tree in which every node can have a maximum of two children is called as Binary Tree.

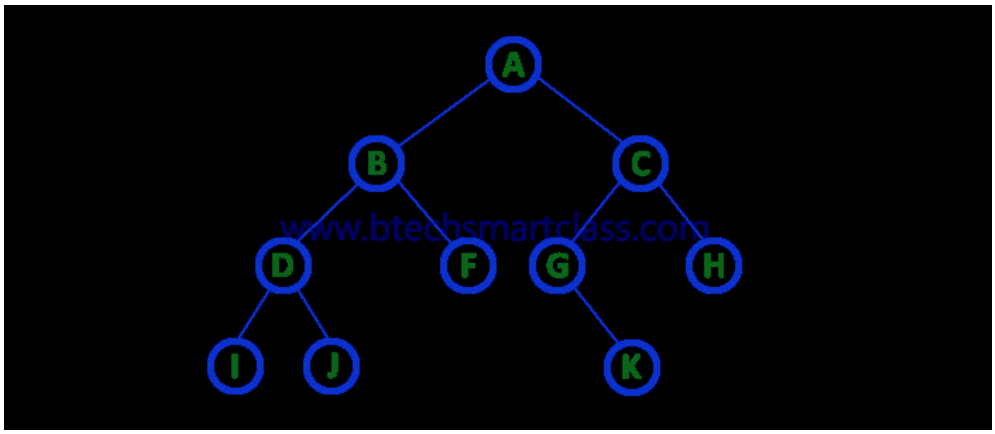
Prerequisite knowledge for Complete understanding and learning of Topic:

- Basic Types in Tree

Detailed content of the Lecture:

- A tree in which every node can have a maximum of two children is called as Binary Tree.
- In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

Example



There are different types of binary trees and they are...

1. Strictly Binary Tree

- In a binary tree, every node can have a maximum of two children.
- But in strictly binary tree, every node should have exactly two children or none.
- That means every internal node must have exactly two children.
- A strictly Binary Tree can be defined as follows...
- A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree.
- Strictly binary tree is also called as Full Binary Tree or Proper Binary Tree or 2-Tree

2. Complete Binary Tree

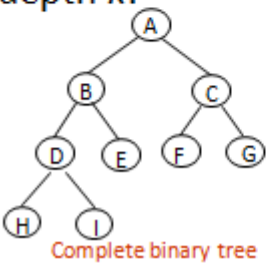
- In a binary tree, every node can have a maximum of two children.
- But in strictly binary tree, every node should have exactly two children or none and in complete

binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be 2 level number of nodes.

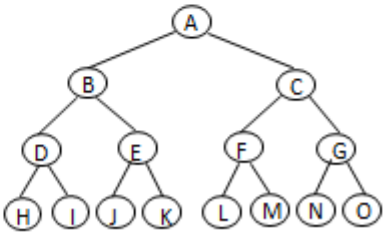
- For example at level 2 there must be 2^2 = 4 nodes and at level 3 there must be 2^3 = 8 nodes.
- Binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree.
- Complete binary tree is also called as Perfect Binary Tree

Full BT VS Complete BT

- A full binary tree of depth *k* is a binary tree of depth *k* having 2^k-1 nodes, *k*>=0.
- A binary tree with *n* nodes and depth *k* is complete *iff* its nodes correspond to the nodes numbered from 1 to *n* in the full binary tree of depth *k*.



Complete binary tree



Full binary tree of depth 4

3. Extended Binary Tree

- A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required.
- The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.

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

- **https://www.javatpoint.com/binary-tree**
- **https://www.geeksforgeeks.org/binary-tree-data-structure/**

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -6 Page No -6.3

Course Faculty

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

LECTURE HANDOUTS

L 13

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **II - Tree Structures**

Date of Lecture: 28.08.2020

Topic of Lecture: Expression trees

Introduction :

- The expression tree is a **tree used to represent the various expressions**.
- The tree data structure is used to represent the expressional statements.

Prerequisite knowledge for Complete understanding and learning of Topic:

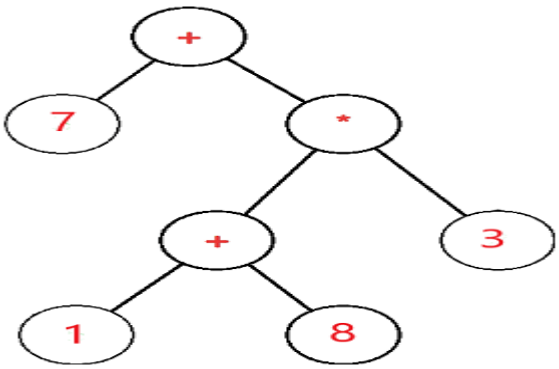
- Non-Linear Data structure
- Basic Types in Tree

Detailed content of the Lecture:

The expression tree is a tree used to represent the various expressions. The tree data structure is used to represent the expressional statements. In this tree, the internal node always denotes the operators.

- The leaf nodes always denote the operands.
- The operations are always performed on these operands.
- The operator present in the depth of the tree is always at the highest priority.
- The operator, which is not much at the depth in the tree, is always at the lowest priority compared to the operators lying at the depth.
- The operand will always present at a depth of the tree; hence it is considered the **highest priority** among all the operators.
- In short, we can summarize it as the value present at a depth of the tree is at the highest priority compared with the other operators present at the top of the tree.
- The main use of these expression trees is that it is used to **evaluate, analyze** and **modify** the various expressions.
- It is also used to find out the associativity of each operator in the expression.
- For example, the + operator is the left-associative and / is the right-associative.
- The dilemma of this associativity has been cleared by using the expression trees.

- These expression trees are formed by using a context-free grammar.
- We have associated a rule in context-free grammars in front of each grammar production.
- These rules are also known as semantic rules, and by using these semantic rules, we can be easily able to construct the expression trees.
- It is one of the major parts of compiler design and belongs to the semantic analysis phase.
- In this semantic analysis, we will use the syntax-directed translations, and in the form of output, we will produce the annotated parse tree.
- An annotated parse tree is nothing but the simple parse associated with the type attribute and each production rule.
- The main objective of using the expression trees is to make complex expressions and can be easily be evaluated using these expression trees.
- It is immutable, and once we have created an expression tree, we can not change it or modify it further.
- To make more modifications, it is required to construct the new expression tree wholly.
- It is also used to solve the postfix, prefix, and infix expression evaluation.
- Expression trees play a very important role in representing the **language-level** code in the form of the data, which is mainly stored in the tree-like structure.
- It is also used in the memory representation of the **lambda** expression. Using the tree data structure, we can express the lambda expression more transparently and explicitly.
- It is first created to convert the code segment onto the data segment so that the expression can easily be evaluated.
- The expression tree is a binary tree in which each external or leaf node corresponds to the operand and each internal or parent node corresponds to the operators so for example expression tree for 7 + ((1+8)*3) would be:



If S is not null, then

If S.value is an operand, then

Return S.value

x = solve(S.left)

y = solve(S.right)

Return calculate(x, y, S.value)

Here in the above example, the expression tree used context-free grammar.

- We have some productions associated with some production rules in this grammar, mainly

known as **semantic rules**. We can define the result-producing from the corresponding production rules using these semantic rules.

- Here we have used the value parameter, which will calculate the result and return it to the grammar's start symbol. S.left will calculate the left child of the node, and similarly, the right child of the node can be calculated using the S.right parameter.

Use of Expression tree

1. The main objective of using the expression trees is to make complex expressions and can be easily be evaluated using these expression trees.
2. It is also used to find out the associativity of each operator in the expression.
3. It is also used to solve the postfix, prefix, and infix expression evaluation.

Implementation of an Expression tree

- To implement the expression tree and write its program, we will be required to use a stack data structure.As we know that the stack is based on the last in first out LIFO principle, the data element pushed recently into the stack has been popped out whenever required.
- For its implementation, the main two operations of the stack, push and pop, are used. Using the push operation, we will push the data element into the stack, and by using the pop operation, we will remove the data element from the stack.

Main functions of the stack in the expression tree implementation:

First of all, we will do scanning of the given expression into left to the right manner, then one by one check the identified character,

1. If a scanned character is an operand, we will apply the push operation and push it into the stack.
2. If a scanned character is an operator, we will apply the pop operation into it to remove the two values from the stack to make them its child, and after then we will push back the current parent node into the stack.

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

- https://www.javatpoint.com/expression-tree-in-data-structure#
- https://www.geeksforgeeks.org/expression-tree/

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -312

Course Faculty

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

LECTURE HANDOUTS

L 14

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **II - Tree Structures**

Date of Lecture: 29.08.2020

Topic of Lecture: Binary tree traversals

Introduction :

- Tree traversal means **visiting each node of the tree**. The tree is a non-linear data structure, and therefore its traversal is different from other linear data structures.
- There is only one way to visit each node/element in linear data structures, i.e. starting from the first value and traversing in a linear order.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Tree
- Types of tree in Non Linear Data structure

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. That is, we cannot 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

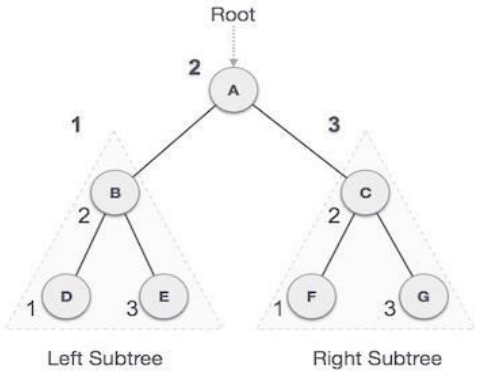
Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We 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.

We 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 −

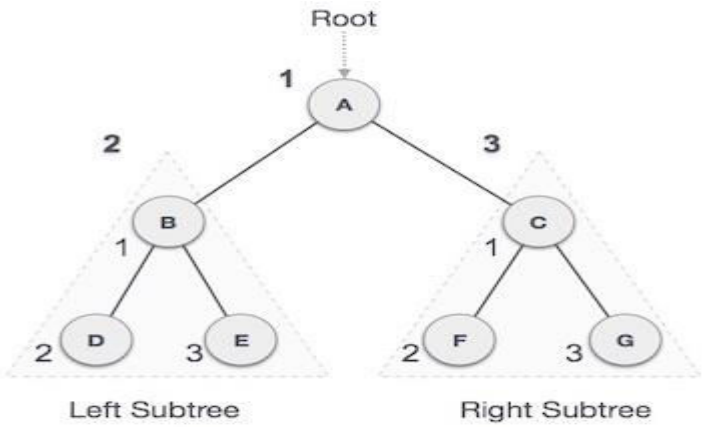
$$D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$$

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.



We start from A, and following pre-order traversal, we 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 −

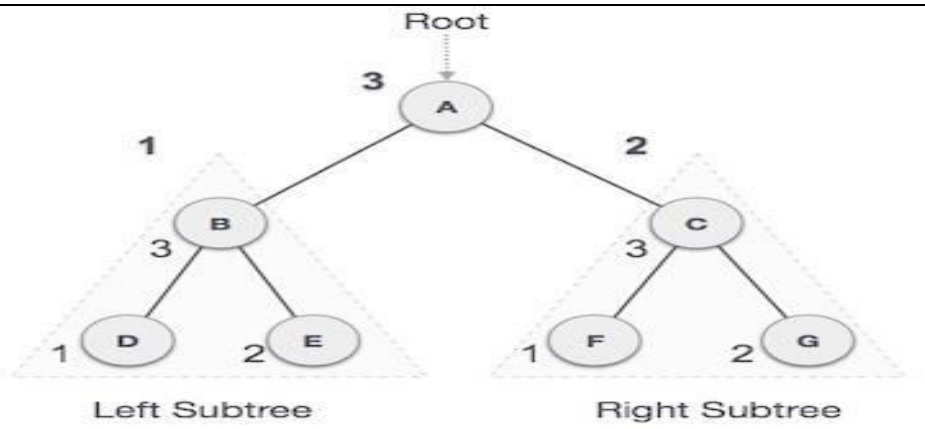
$$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$$

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 we traverse the left subtree, then the right subtree and finally the root node.



We start from A, and following Post-order traversal, we 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.:

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -277

Course Faculty

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

LECTURE HANDOUTS

L15

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **II - Tree Structures**

Date of Lecture: 30.08.2020

Topic of Lecture: left child right sibling data structures for general trees

Introduction :

- Left-Child Right-Sibling Representation is a different representation of an n-ary tree where instead of holding a reference to each and every child node, a node holds just two references, first a reference to it's first child, and the other to it's immediate next sibling.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Tree
- Types of tree in Non Linear Data structure

Detailed content of the Lecture:

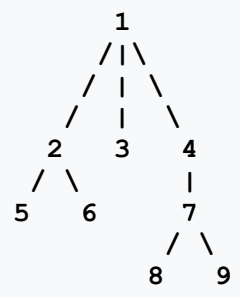
- This new transformation not only removes the need of advance knowledge of the number of children a node has, but also limits the number of references to a maximum of two, thereby making it so much easier to code.

Examples:

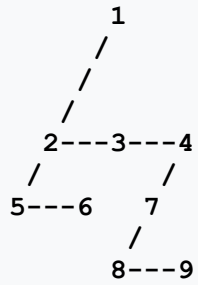
```
Left Child Right Sibling tree representation
      10
      |
      2 -> 3 -> 4 -> 5
      |    |
      6    7 -> 8 -> 9
```

Doubly chained trees

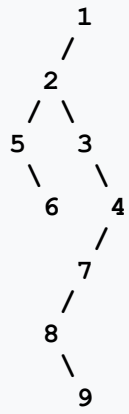
Processing a k-ary tree to LC-RS binary tree, every node is linked and aligned with the left child, and the next nearest is a sibling. For example, we have a ternary tree below:



We can re-write it by putting the left child node to one level below its parents and by putting the sibling next to the child at the same level – it will be linear (same line).



We can transform this tree to a binary tree by turning each sibling 45° clockwise.[6]



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

- **https://www.geeksforgeeks.org/left-child-right-sibling-representation-tree/**
- **https://tutorialspoint.dev/data-structure/binary-tree-data-structure/creating-tree-left-child-right-sibling-representation**

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -261

Course Faculty

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

L16

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **II - Tree Structures**

Date of Lecture: 01.09.2020

Topic of Lecture: Applications of trees

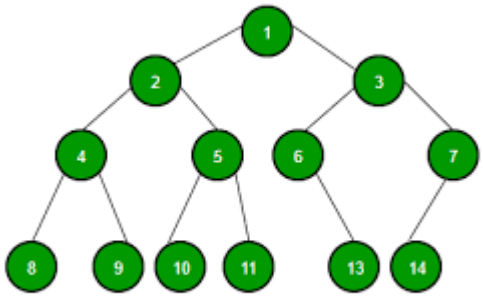
Introduction :

- **A tree** is also one of the data structures that represent hierarchical data
  - A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Tree
- Types of Trees

Detailed content of the Lecture:



- .

One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

file system

```
    /   <-- root
  /
...       home
     /
  ugrad       course
   /        /   |
  ...       cs101 cs112 cs113
```

- If we organize keys in form of a tree (with some ordering e.g., BST), we can search for a given

key in moderate time (quicker than Linked List and slower than arrays).

- Self-balancing search trees like AVL and Red-Black trees guarantee an upper bound of O(Logn) for search.
- We can insert/delete keys in moderate time (quicker than Arrays and slower than Unordered Linked Lists).
- Self-balancing search trees like AVL and Red-Black trees guarantee an upper bound of O(Logn) for insertion/deletion.Like Linked Lists and unlike Arrays, Pointer implementation of trees don't have an upper limit on number of nodes as nodes are linked using pointers.

Other Applications :

- Heap is a tree data structure which is implemented using arrays and used to implement priority queues.
- B-Tree and B+ Tree : They are used to implement indexing in databases.
- Syntax Tree: Used in Compilers.
- K-D Tree: A space partitioning tree used to organize points in K dimensional space.
- Trie : Used to implement dictionaries with prefix lookup.
- Suffix Tree : For quick pattern searching in a fixed text.

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

- **https://www.geeksforgeeks.org/applications-of-tree-data-structure/**
- **https://tutorialspoint.dev/data-structure/binary-tree-data-structure/applications-of-tree-data-structure**

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -305

Course Faculty

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

LECTURE HANDOUTS

L17

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **II - Tree Structures**

Date of Lecture: 03.09.2020

Topic of Lecture: Huffman Algorithm

Introduction :

- Huffman coding is a lossless data compression algorithm.
- The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.
- The most frequent character gets the smallest code and the least frequent character gets the largest code.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Huffman Coding

Detailed content of the Lecture:

- The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character.
- This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.
- Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1.
- This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b.
- If the compressed bit stream is 0001, the de-compressed output may be "**cccd" or "ccb" or "acd" or "ab".**

Applications of Huffman Coding.

There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.
2. Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

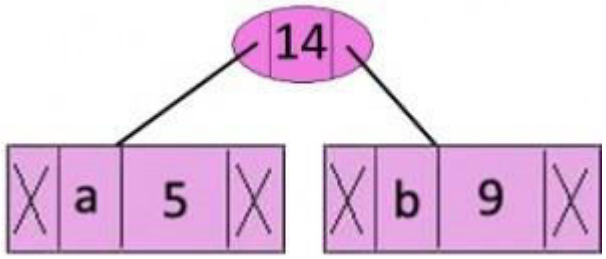
2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

5. Let us understand the algorithm with an example:

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

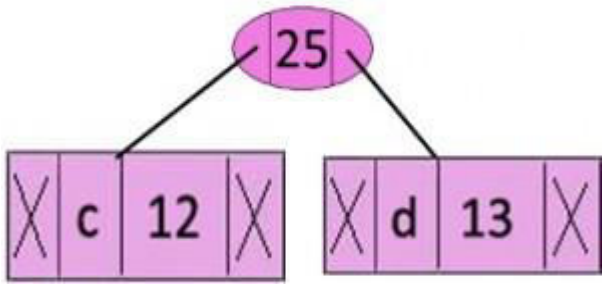
Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14.



Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25



- Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes

character Frequency

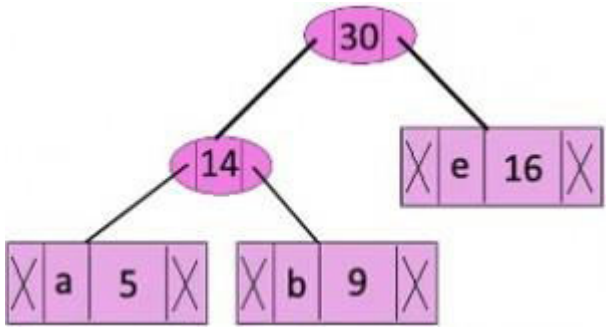
Internal Node 14

e 16

Internal Node 25

f 45

Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30



Now min heap contains 3 nodes.

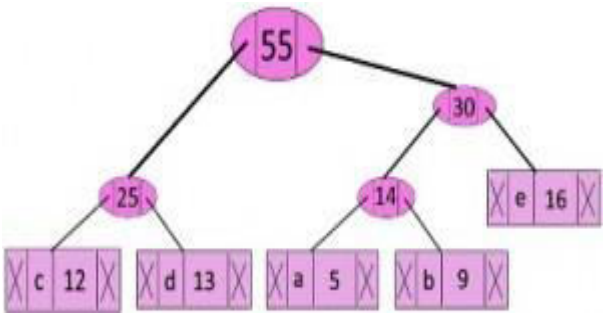
character Frequency

Internal Node 25

Internal Node 30

f 45

Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55



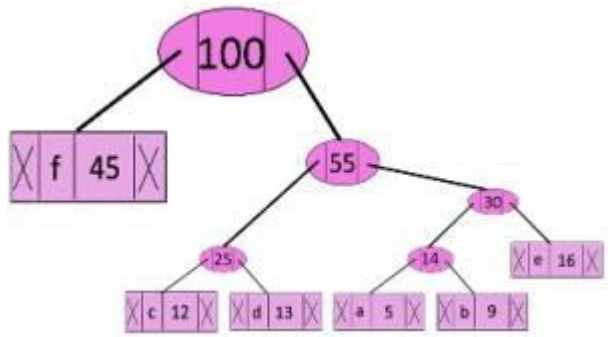
Now min heap contains 2 nodes.

character Frequency

f 45

Internal Node 55

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100



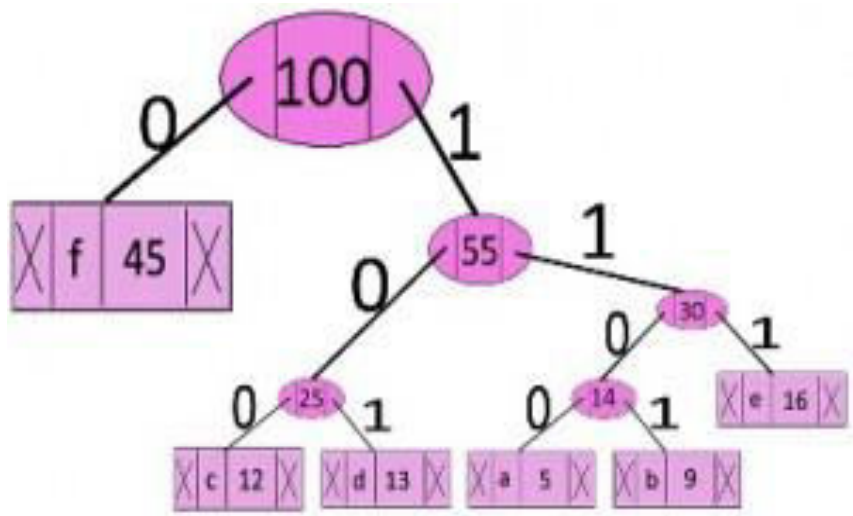
Now min heap contains only one node.

character Frequency

Internal Node 100

Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:
Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



The codes are as follows:

character code-word

f	0
c	100
d	101
a	1100
b	1101
e	111

Video Content / Details of website for further learning (if any): • https://www.programiz.com/dsa/huffman-coding • https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/
Important Books/Journals for further learning including the page nos.: • Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -287

Course Faculty

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 - 18

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **II - Tree Structures**

Date of Lecture: 04.09.2020

Topic of Lecture: Binary search tree

Introduction : (Maximum 5 sentences)

- A binary search tree (BST), also called an ordered or sorted binary tree, is a **rooted binary tree data structure** whose internal nodes each store a key greater than all the keys in the node's left subtree and less than those in its right subtree.

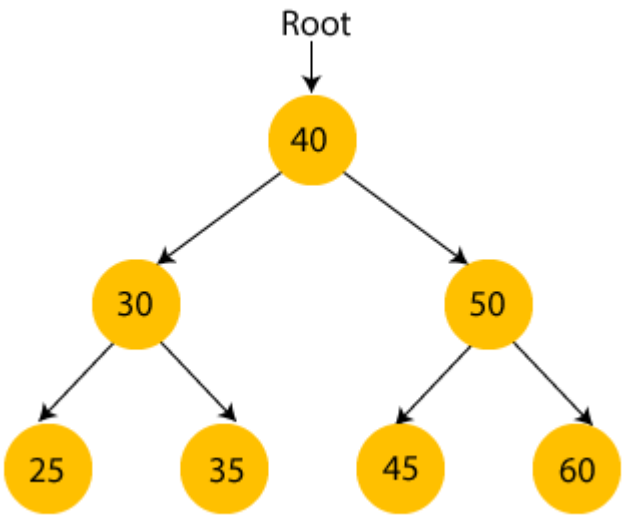
Prerequisite knowledge for Complete understanding and learning of Topic:

- Types of Tree
- Binary Tree

Detailed content of the Lecture:

A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.

Let's understand the concept of Binary search tree with an example.

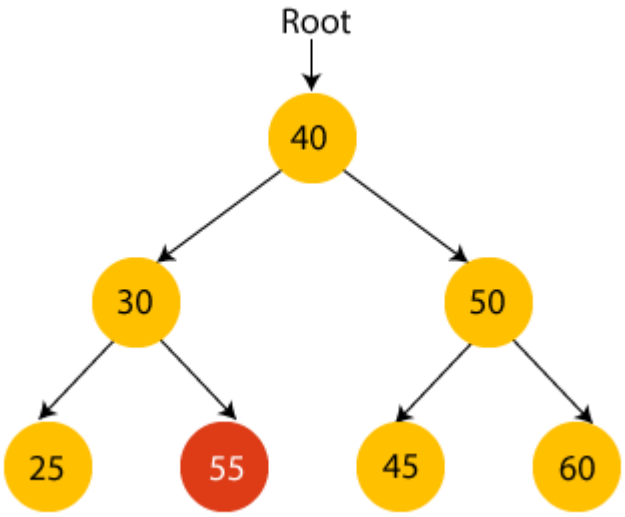


In the above figure, we can observe that the root node is 40, and all the nodes of the left subtree are smaller than the root node, and all the nodes of the right subtree are greater than the root node.

Similarly, we can see the left child of root node is greater than its left child and smaller than its right child. So, it also satisfies the property of binary search tree. Therefore, we can say that

the tree in the above image is a binary search tree.

Suppose if we change the value of node 35 to 55 in the above tree, check whether the tree will be binary search tree or not.



In the above tree, the value of root node is 40, which is greater than its left child 30 but smaller than right child of 30, i.e., 55. So, the above tree does not satisfy the property of Binary search tree. Therefore, the above tree is not a binary search tree.

Advantages of Binary search tree

- Searching an element in the Binary search tree is easy as we always have a hint that which subtree has the desired element.
- As compared to array and linked lists, insertion and deletion operations are faster in BST.

Example of creating a binary search tree

Now, let's see the creation of binary search tree using an example.

Suppose the data elements are **- 45, 15, 79, 90, 10, 55, 12, 20, 50**

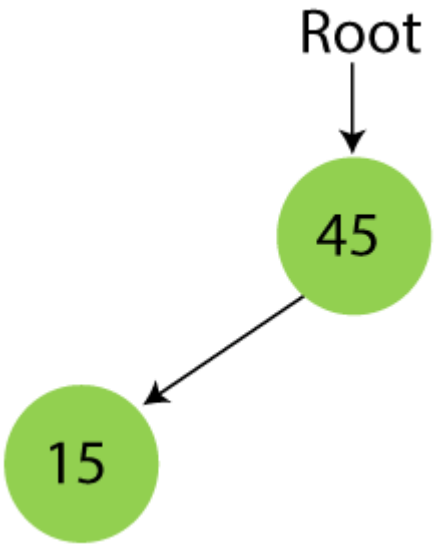
- First, we have to insert **45** into the tree as the root of the tree.
- Then, read the next element; if it is smaller than the root node, insert it as the root of the left subtree, and move to the next element.
- Otherwise, if the element is larger than the root node, then insert it as the root of the right subtree.

Now, let's see the process of creating the Binary search tree using the given data element. The process of creating the BST is shown below -

Step 1 - Insert 45.

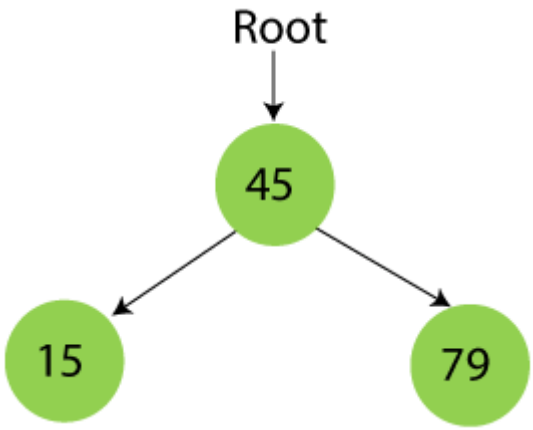
Step 2 - Insert 15.

As 15 is smaller than 45, so insert it as the root node of the left subtree.



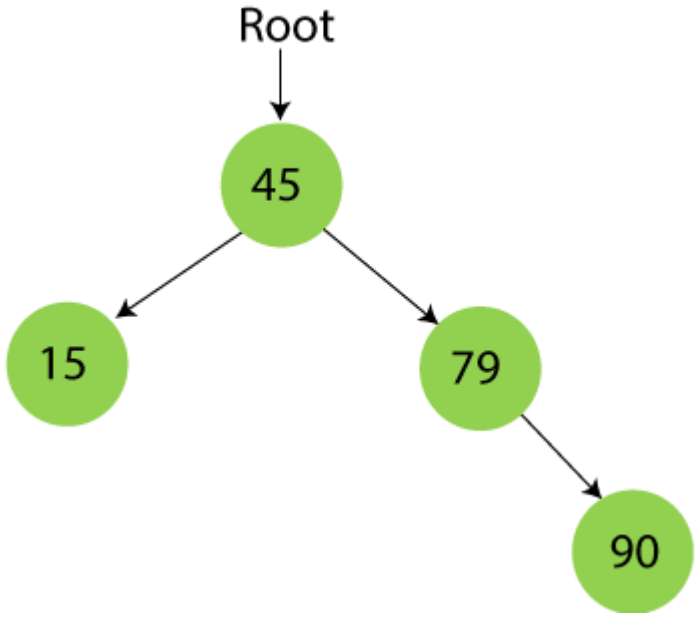
Step 3 - Insert 79.

As 79 is greater than 45, so insert it as the root node of the right subtree.



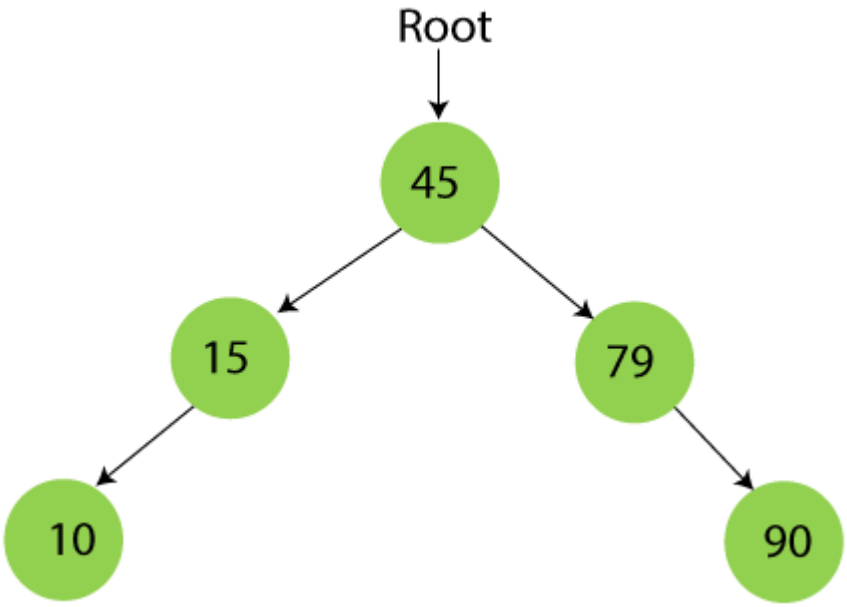
Step 4 - Insert 90.

90 is greater than 45 and 79, so it will be inserted as the right subtree of 79.



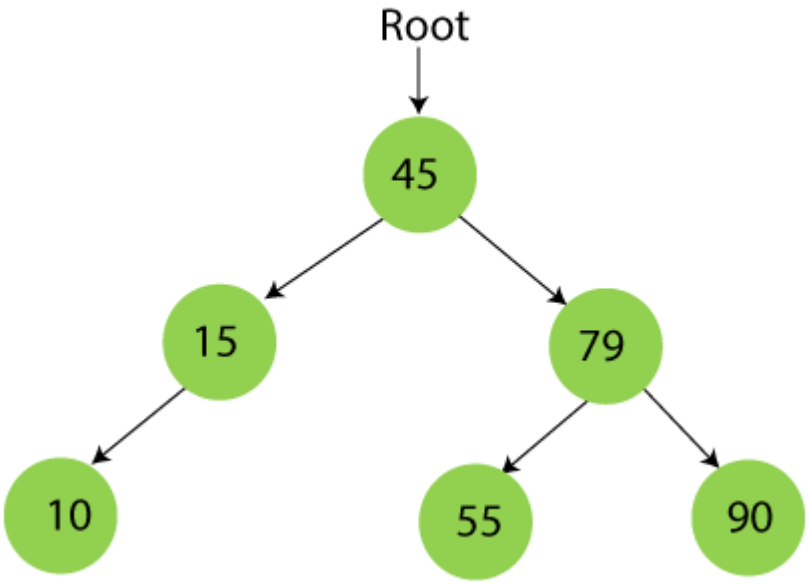
Step 5 - Insert 10.

10 is smaller than 45 and 15, so it will be inserted as a left subtree of 15.



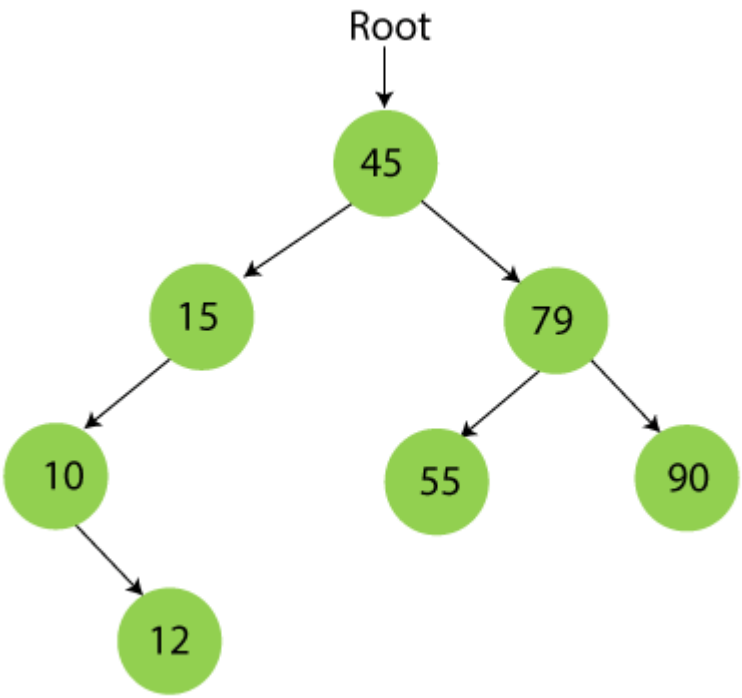
Step 6 - Insert 55.

55 is larger than 45 and smaller than 79, so it will be inserted as the left subtree of 79.



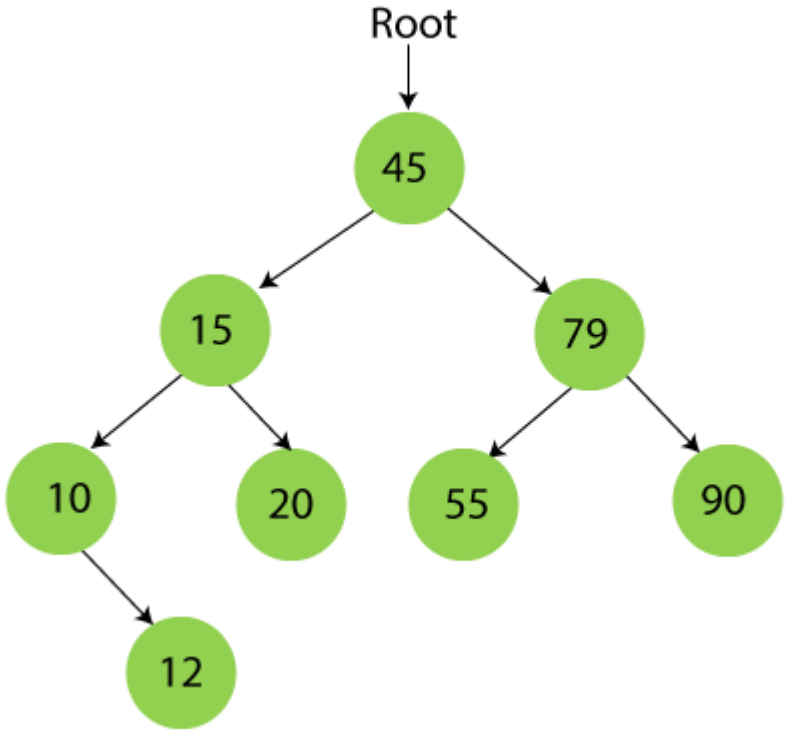
Step 7 - Insert 12.

12 is smaller than 45 and 15 but greater than 10, so it will be inserted as the right subtree of 10.



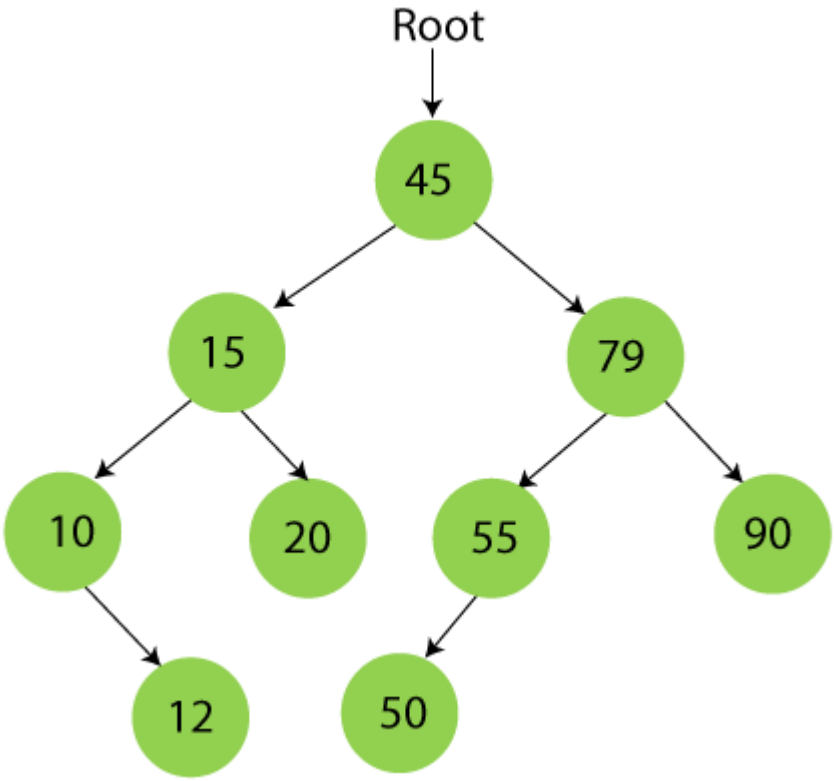
Step 8 - Insert 20.

20 is smaller than 45 but greater than 15, so it will be inserted as the right subtree of 15.



Step 9 - Insert 50.

50 is greater than 45 but smaller than 79 and 55. So, it will be inserted as a left subtree of 55.



Now, the creation of binary search tree is completed. After that, let's move towards the operations that can be performed on Binary search tree.

We can perform insert, delete and search operations on the binary search tree.

Let's understand how a search is performed on a binary search tree.

Searching in Binary search tree

Searching means to find or locate a specific element or node in a data structure. In Binary search tree, searching a node is easy because elements in BST are stored in a specific order. The steps of searching a node in Binary Search tree are listed as follows -

1. First, compare the element to be searched with the root element of the tree.
2. If root is matched with the target element, then return the node's location.
3. If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.
4. If it is larger than the root element, then move to the right subtree.
5. Repeat the above procedure recursively until the match is found.
6. If the element is not found or not present in the tree, then return NULL.

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

- **https://www.geeksforgeeks.org/binary-search-tree-data-structure/**
- **https://www.javatpoint.com/binary-search-tree**
-

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -258

Course Faculty

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 19**

MCA | **I / I**

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **III - Graphs**

Date of Lecture: 07.09.2020

Topic of Lecture: Definitions , Representation of graph

Introduction :

- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links.
- The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.

Prerequisite knowledge for Complete understanding and learning of Topic:

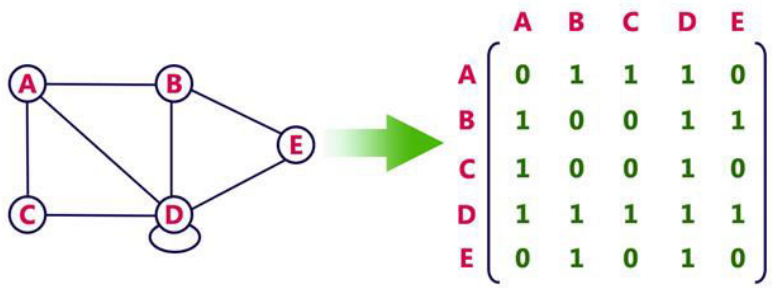
- Types of Non-Linear Structure
- Basics of Graph

Detailed content of the Lecture:

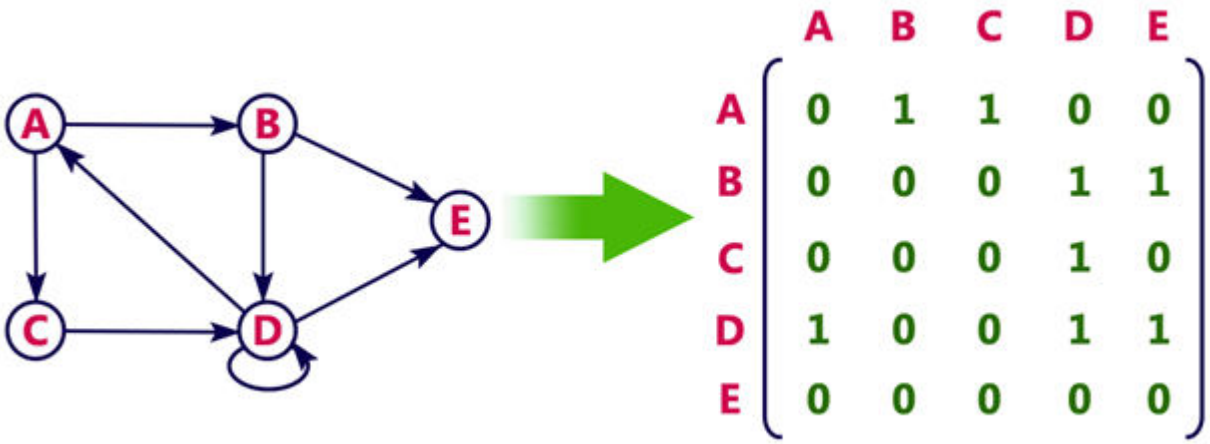
- In graph theory, a graph representation is a technique to store graph into the memory of computer.
- To represent a graph, we just need the set of vertices, and for each vertex the neighbors of the vertex (vertices which is directly connected to it by an edge).
- If it is a weighted graph, then the weight will be associated with each edge.
- There are different ways to optimally represent a graph, depending on the density of its edges, type of operations to be performed and ease of use.

- 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, ai,j = 1, otherwise ai,j= 0
- Example
- Consider the following **undirected graph representation**:

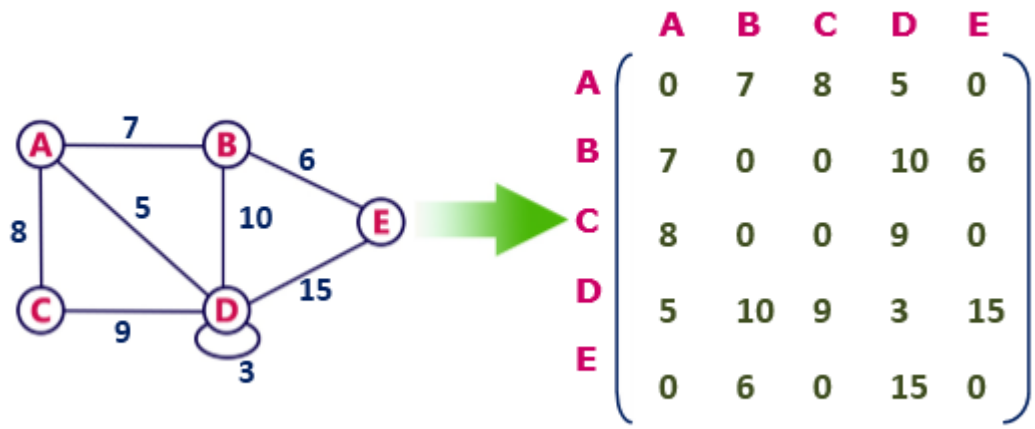
- **Undirected graph representation**



- **Directed graph representation**

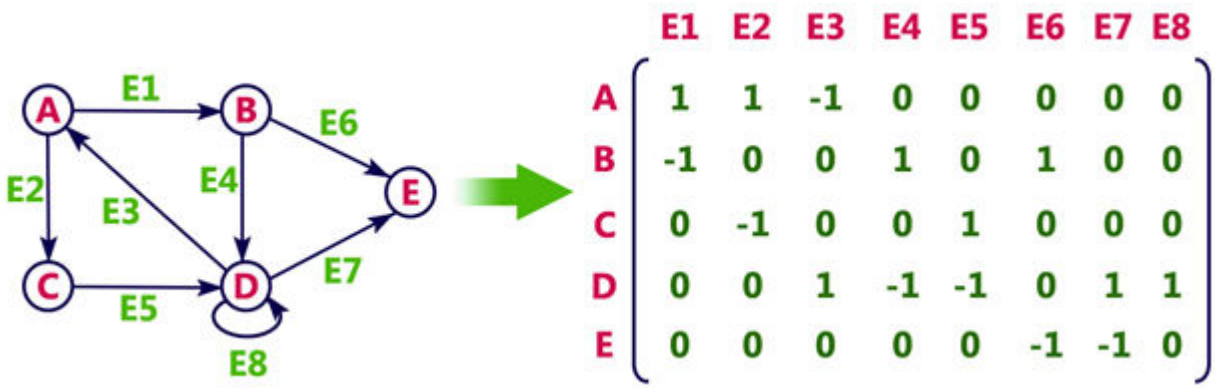


- **Undirected weighted graph represenation**



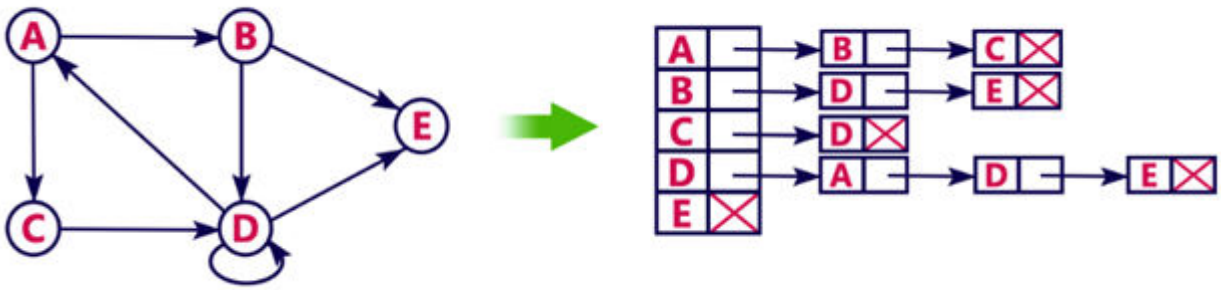
- Incidence Matrix
- In **Incidence matrix representation**, graph can be represented using a matrix of size:
- Total number of vertices by total number of edges.
- It means if a graph has 4 vertices and 6 edges, then it can be represented using a matrix of 4X6 class. In this matrix, columns represent edges and rows represent vertices.
- This matrix is filled with either **0 or 1** or -1. Where,
- 0 is used to represent row edge which is not connected to column vertex.
- 1 is used to represent row edge which is connected as outgoing edge to column vertex.
- -1 is used to represent row edge which is connected as incoming edge to column vertex.

- Example
- Consider the following directed graph representation.

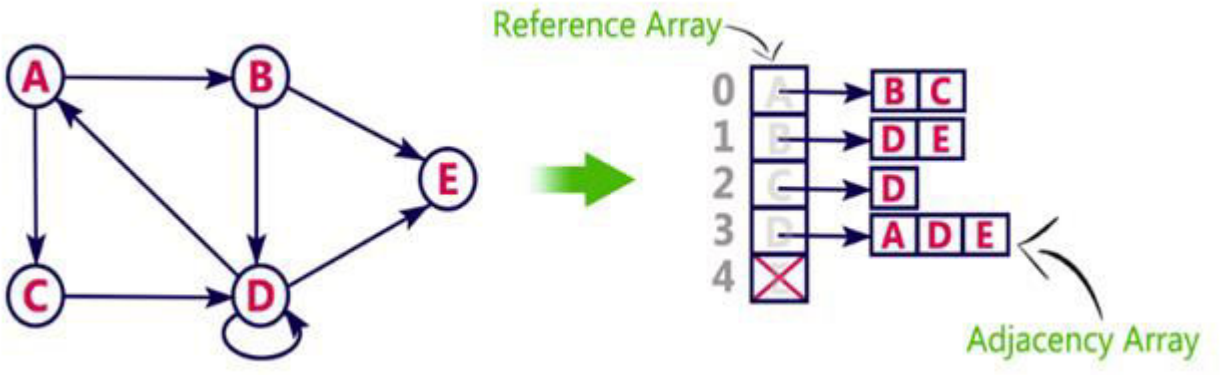


-
-

- 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.
- We have 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:



-
- We can also implement this representation using array as follows:



-
- **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.
- **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 website for further learning (if any):

- **https://www.javatpoint.com/graph-theory-graph-representations**
- **https://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.html**
- **https://www.geeksforgeeks.org/graph-and-its-representations/**

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -520

Course Faculty

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

LECTURE HANDOUTS

L 20

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **III - Graphs**

Date of Lecture: 08.09.2020

Topic of Lecture: Graph Traversal - Depth-first traversal

Introduction : (Maximum 5 sentences)

- In computer science, graph traversal refers to the process of visiting (checking and/or updating) each vertex in a graph.
- Graph traversal is a technique used for a searching vertex in a graph.
- A graph traversal finds the edges to be used in the search process without creating loops.

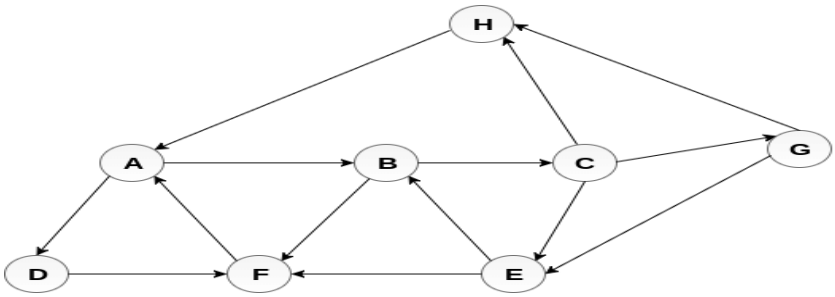
Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Graph

Detailed content of the Lecture:

- Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until we find the goal node or the node which has no children. The algorithm, then backtracks from the dead end towards the most recent node that is yet to be completely unexplored.
- The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.
- Algorithm
- **Step 1:** SET STATUS = 1 (ready state) for each node in G
- **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until STACK is empty
- **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
- **Step 5:** Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) **and set their**
  STATUS = 2 (waiting state)
  [END OF LOOP]
- **Step 6:** EXIT
- Example :

- Consider the graph G along with its adjacency list, given in the figure below. Calculate the order to print all the nodes of the graph starting from node H, by using depth first search (DFS) algorithm.



Adjacency Lists

A : B, D
B : C, F
C : E, G, H
G : E, H
E : B, F
F : A
D : F
H : A

Solution :

- Push H onto the stack
- STACK : H
- POP the top element of the stack i.e. H, print it and push all the neighbours of H onto the stack that are is ready state.
- Print H
- STACK : A
- Pop the top element of the stack i.e. A, print it and push all the neighbours of A onto the stack that are in ready state.
- Print A
- Stack : B, D
- Pop the top element of the stack i.e. D, print it and push all the neighbours of D onto the stack that are in ready state.
- Print D
- Stack : B, F
- Pop the top element of the stack i.e. F, print it and push all the neighbours of F onto the stack that are in ready state.
- Print F
- Stack : B
- Pop the top of the stack i.e. B and push all the neighbours

- Print B
- Stack : C
- Pop the top of the stack i.e. C and push all the neighbours.

- Print C
- Stack : E, G
- Pop the top of the stack i.e. G and push all its neighbours.

- Print G
- Stack : E
- Pop the top of the stack i.e. E and push all its neighbours.

- Print E
- Stack :
- Hence, the stack now becomes empty and all the nodes of the graph have been traversed.
- The printing sequence of the graph will be :
- H → A → D → F → B → C → G → E

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

- **https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/GraphTraversal.html**
- **http://www.btechsmartclass.com/data_structures/graph-traversal-dfs.html**

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -560

Course Faculty

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

L 21

LECTURE HANDOUTS

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **III-Graph**

Date of Lecture: 10.09.2020

Topic of Lecture: Breadth-first traversal.

- **Introduction : (Maximum 5 sentences)**
- Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbouring nodes.
- Then, it selects the nearest node and explore all the unexplored nodes.
- The algorithm follows the same process for each of the nearest node until it finds the goal..

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basic of Graph

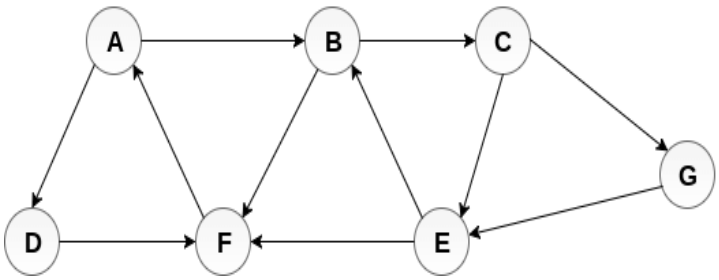
Detailed content of the Lecture:

- The algorithm of breadth first search is given below. The algorithm starts with examining the n all of its neighbours. In the next step, the neighbours of the nearest node of A are explored a continues in the further steps. The algorithm explores all neighbours of all the nodes and ensure node is visited exactly once and no node is visited twice.
- **Algorithm**
- **Step 1:** SET STATUS = 1 (ready state)
  for each node in G
- **Step 2:** Enqueue the starting node A
  and set its STATUS = 2
  (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until
  QUEUE is empty
- **Step 4:** Dequeue a node N. Process it
  and set its STATUS = 3
  (processed state).

- **Step 5:** Enqueue all the neighbours of
  N that are in the ready state
  (whose STATUS = 1) and set
  their STATUS = 2

  (waiting state)
  [END OF LOOP]

- **Step 6:** EXIT
- Example
- Consider the graph G shown in the following image, calculate the minimum path p from node A to node
  Given that each edge has a length of 1.
-



Adjacency Lists

A : B, D
B : C, F
C : E, G
G : E
E : B, F
F : A
D : F

Solution:

- Minimum Path P can be found by applying breadth first search algorithm that will begin at node A and w
  end at E. the algorithm uses two queues, namely **QUEUE1** and **QUEUE2**. **QUEUE1** holds all the noc
  that are to be processed while **QUEUE2** holds all the nodes that are processed and deleted from **QUEUE**
- **Lets start examining the graph from Node A.**
- Add A to QUEUE1 and NULL to QUEUE2.
- QUEUE1 = {A}
- QUEUE2 = {NULL}
- Delete the Node A from QUEUE1 and insert all its neighbours. Insert Node A into QUEUE2
- QUEUE1 = {B, D}
- QUEUE2 = {A}
- Delete the node B from QUEUE1 and insert all its neighbours. Insert node B into QUEUE2.
- QUEUE1 = {D, C, F}
- QUEUE2 = {A, B}
- Delete the node D from QUEUE1 and insert all its neighbours. Since F is the only
- neighbour of it which has been inserted, we will not insert it again. Insert node D into QUEUE2.
- QUEUE1 = {C, F}
- QUEUE2 = { A, B, D}

- Delete the node C from QUEUE1 and insert all its neighbours. Add node C to QUEUE2.
- QUEUE1 = {F, E, G}
- QUEUE2 = {A, B, D, C}
- Remove F from QUEUE1 and add all its neighbours. Since all of its neighbours has already been adde we will not add them again. Add node F to QUEUE2.
- QUEUE1 = {E, G}
- QUEUE2 = {A, B, D, C, F}
- Remove E from QUEUE1, all of E's neighbours has already been added to QUEUE1 therefore
- we will not add them again. All the nodes are visited and the target node i.e. E is encountered in QUEUE2.
- QUEUE1 = {G}
- QUEUE2 = {A, B, D, C, F, E}
- Now, backtrack from E to A, using the nodes available in QUEUE2.
- The minimum path will be **A → B → C → E**.

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

- **https://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm**
- **https://www.javatpoint.com/breadth-first-search-algorithm**

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -573

Course Faculty

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 22

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **III-Graphs**

Date of Lecture: 21.09.2020

Topic of Lecture: Applications of graphs Topological sort

Introduction :

- Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

Prerequisite knowledge for Complete understanding and learning of Topic:

- **Basics of Graph.**

Detailed content of the Lecture:

1. Computer Science

- In computer science graph theory is used for the study of algorithms like:
- Dijkstra's Algorithm
- Prims's Algorithm
- Kruskal's Algorithm
- Graphs are used to define the flow of computation.
- Graphs are used to represent networks of communication.
- Graphs are used to represent data organization.
- Graph transformation systems work on rule-based in-memory manipulation of graphs. Graph databases ensure transaction-safe, persistent storing and querying of graph structured data.
- Graph theory is used to find shortest path in road or a network.

- In Google Maps, various locations are represented as vertices or nodes and the roads are represented as edges and graph theory is used to find the shortest path between two nodes.

2. Electrical Engineering

- In Electrical Engineering, graph theory is used in designing of circuit connections. These circuit connections are named as topologies. Some topologies are series, bridge, star and parallel topologies.

3. Linguistics

- In linguistics, graphs are mostly used for parsing of a language tree and grammar of a language tree.
- Semantics networks are used within lexical semantics, especially as applied to computers, modeling word meaning is easier when a given word is understood in terms of related words.
- Methods in phonology (e.g. theory of optimality, which uses lattice graphs) and morphology (e.g. morphology of finite - state, using finite-state transducers) are common in the analysis of language as a graph.

4. Physics and Chemistry

- In physics and chemistry, graph theory is used to study molecules.
- The 3D structure of complicated simulated atomic structures can be studied quantitatively by gathering statistics on graph-theoretic properties related to the topology of the atoms.
- Statistical physics also uses graphs. In this field graphs can represent local connections between interacting parts of a system, as well as the dynamics of a physical process on such systems.
- Graphs are also used to express the micro-scale channels of porous media, in which the vertices represent the pores and the edges represent the smaller channels connecting the pores.
- Graph is also helpful in constructing the molecular structure as well as lattice of the molecule. It also helps us to show the bond relation in between atoms and molecules, also help in comparing structure of one molecule to other.

5. Computer Network

- In computer network, the relationships among interconnected computers within the network, follow the principles of graph theory.
- Graph theory is also used in network security.
- We can use the vertex coloring algorithm to find a proper coloring of the map with four colors.
- Vertex coloring algorithm may be used for assigning at most four different frequencies for any GSM (Grouped Special Mobile) mobile phone networks.

6. Social Sciences

- Graph theory is also used in sociology. For example, to explore rumor spreading, or to measure

actors' prestige notably through the use of social network analysis software.

- Acquaintanceship and friendship graphs describe whether people know each other or not.
- In influence graphs model, certain people can influence the behavior of others.
- In collaboration graphs model to check whether two people work together in a particular way, such as acting in a movie together.

7. Biology

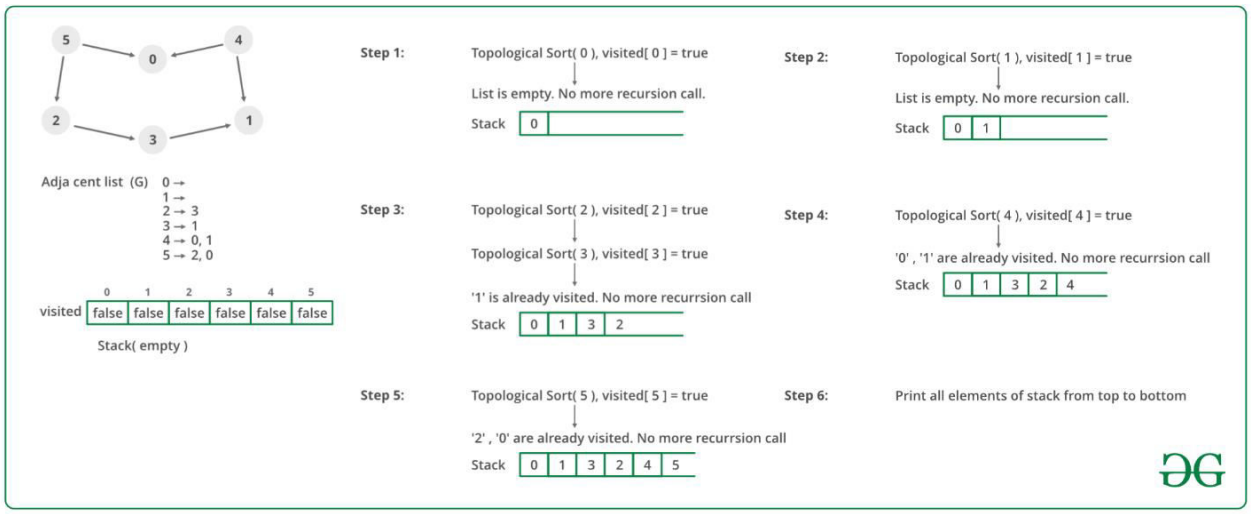
- Nodes in biological networks represent bimolecular such as genes, proteins or metabolites, and edges connecting these nodes indicate functional, physical or chemical interactions between the corresponding bimolecular.
- Graph theory is used in transcriptional regulation networks.
- It is also used in Metabolic networks.
- In PPI (Protein - Protein interaction) networks graph theory is also useful.
- Characterizing drug - drug target relationships.
- 8. Mathematics
- In mathematics, operational research is the important field. Graph theory provides many useful applications in operational research. Like:
- Minimum cost path.
- A scheduling problem.
- 9. General
- Graphs are used to represent the routes between the cities. With the help of tree that is a type of graph, we can create hierarchical ordered information such as family tree.

TOPOLOGICAL SORTING

- Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering.
- Topological Sorting for a graph is not possible if the graph is not a DAG.
- For example, a topological sorting of the following graph is “5 4 2 3 1 0”. There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is “4 5 2 3 1 0”. The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no incoming edges).
- Algorithm to find Topological Sorting:
- We recommend to first see the implementation of DFS. We can modify DFS to find Topological

Sorting of a graph.

- In DFS, we start from a vertex, we first print it and then recursively call DFS for its adjacent vertices. In topological sorting, we use a temporary stack.
- We don't print the vertex immediately, we first recursively call topological sorting for all its adjacent vertices, then push it to a stack. Finally, print contents of the stackBelow image is an illustration of the above approach:



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

- https://www.javatpoint.com/graph-theory-applications
- https://www.geeksforgeeks.org/topological-sorting/

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -517

Course Faculty

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 23

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **III-Graph.**

Date of Lecture: 22.09.2020

Topic of Lecture: Shortest-path algorithms

Introduction :

- The problem of finding the shortest path in a graph from one vertex to another. "Shortest" may be least number of edges, least total weight, etc.
- Also known as single-pair shortest-path problem.
- In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Graph.

Detailed content of the Lecture:

- Den computer networks, the shortest path algorithms aim to find the optimal paths between the network nodes so that routing cost is minimized. They are direct applications of the shortest path algorithms proposed in graph theory.

- Explanation
- Consider that a network comprises of N vertices (nodes or network devices) that are connected by M edges (transmission lines).
- Each edge is associated with a weight, representing the physical distance or the transmission delay of the transmission line.

- Common Shortest Path Algorithms
- Some common shortest path algorithms are −

Bellman Ford's Algorithm

- Dijkstra's Algorithm

- Floyd Warshall's Algorithm

- *The following sections describes each of these algorithms.*

- *Bellman Ford Algorithm*
- Input − A graph representing the network; and a source node, s

- Output − Shortest path from s to all other nodes.
- Initialize distances from s to all nodes as infinite (∞); distance to itself as 0; an array dist[] of size |V| (number of nodes) with all values as ∞ except dist[s].

- Calculate the shortest distances iteratively.
- Repeat |V|- 1 times for each node except s −

- Repeat for each edge connecting vertices u and v −

- If dist[v] > (dist[u] + weight of edge u-v), Then

- Update dist[v] = dist[u] + weight of edge u-v

- The array dist[] contains the shortest path from s to every other node.

- *Dijkstra's Algorithm*
- Input − A graph representing the network; and a source node, s

- Output − A shortest path tree, spt[], with s as the root node.

- Initializations −
- An array of distances dist[] of size |V| (number of nodes), where dist[s] = 0 and dist[u] = ∞ (infinite), where u represents a node in the graph except s.

- An array, Q, containing all nodes in the graph. When the algorithm runs into completion, Q will become empty.

- An empty set, S, to which the visited nodes will be added. When the algorithm runs into completion, S will contain all the nodes in the graphRepeat while Q is not empty −

- Remove from Q, the node, u having the smallest dist[u] and which is not in S. In the first run, dist[s] is removed.

- Add u to S, marking u as visited.

- For each node v which is adjacent to u, update dist[v] as −
- If (dist[u] + weight of edge u-v) < dist[v], Then
- Update dist[v] = dist[u] + weight of edge u-v
- The array dist[] contains the shortest path from s to every other node.

Floyd Warshall Algorithm

- Input − A cost adjacency matrix, adj[][], representing the paths between the nodes in the network.
- Output − A shortest path cost matrix, cost[][], showing the shortest paths in terms of cost between each pair of nodes in the graph.
- Populate cost[][] as follows:
- If adj[][] is empty Then cost[][] = ∞ (infinite)
- Else cost[][] = adj[][]
- N = |V|, where V represents the set of nodes in the network.
- Repeat for k = 1 to N –Repeat for i = 1 to N −
- Repeat for j = 1 to N −
- If cost[i][k] + cost[k][j] < cost[i][j], Then
- Update cost[i][j] := cost[i][k] + cost[k][j]
- The matrix cost[][] contains the shortest cost from each node, i , to every other node, j.

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

- https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/https://brilliant.org/wiki/dijkstras-short-path-finder/

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

- Data Structures and Algorithm Analysis in C++", M. A. Weiss , Pearson Education Asia,2013 Page No-148

Course Faculty

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

L24

LECTURE HANDOUTS

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **III – Graph.**

Date of Lecture: 23.09.2020

Topic of Lecture: Minimum Spanning tree

Introduction :

- A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of General Tree

Detailed content of the Lecture:

- The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees.
- Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.
- Minimum spanning tree has direct application in the design of networks.
- It is used in algorithms approximating the travelling salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching. Other practical applications are:
- Cluster Analysis
- Handwriting recognition
- Image segmentation

There are two famous algorithms for finding the Minimum Spanning Tree:

Kruskal's Algorithm

- Kruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge

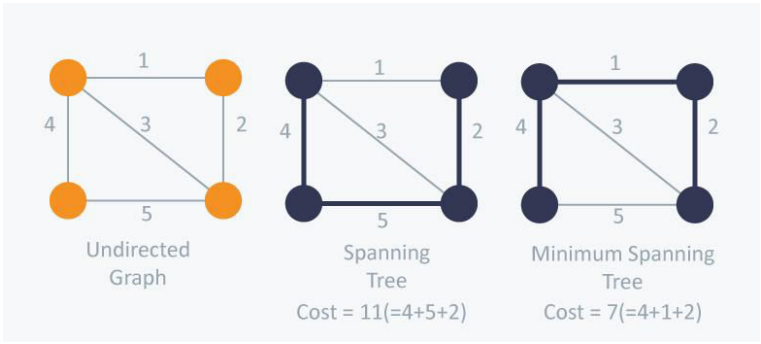
which has least weight and add it to the growing spanning tree.

Algorithm Steps:

- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.

So now the question is how to check if 2 vertices are connected or not ?
This could be done using DFS which starts from the first vertex, then check if the second vertex is visited or not. But DFS will make time complexity large as it has an order of O(V+E) where V is the number of vertices, E is the number of edges. So the best solution is **"Disjoint Sets":**
Disjoint sets are sets whose intersection is the empty set so it means that they don't have any element in common.

Consider following example:

- 

There are two famous algorithms for finding the Minimum Spanning Tree:

Kruskal's Algorithm

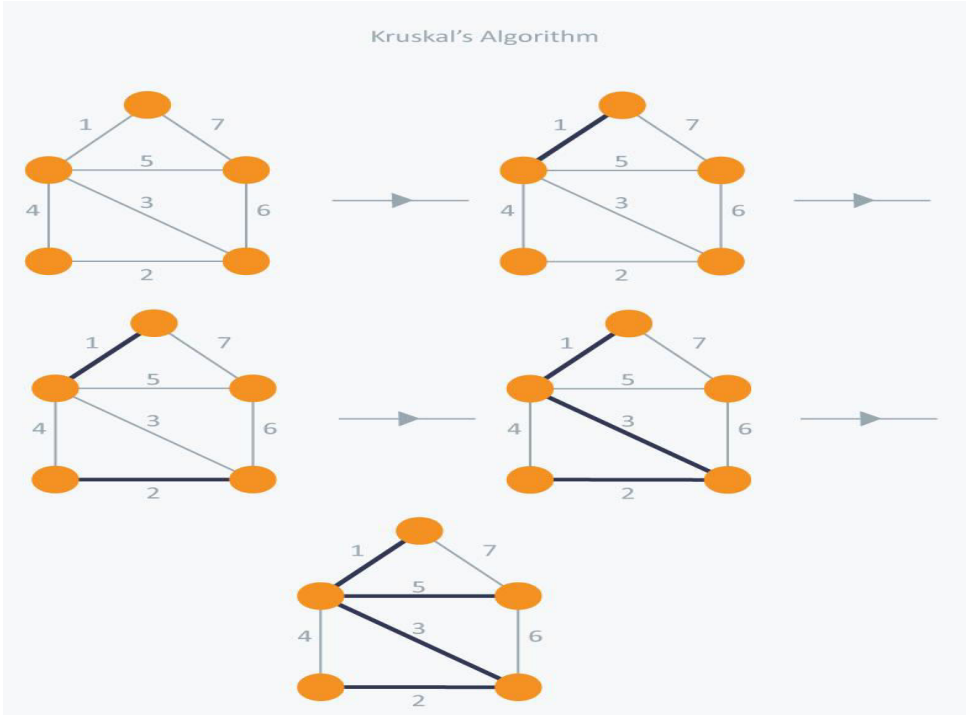
Kruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.

Algorithm Steps:

- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.

So now the question is how to check if 2 vertices are connected or not ?
This could be done using DFS which starts from the first vertex, then check if the second vertex is visited or not. But DFS will make time complexity large as it has an order of O(V+E) where V is the number of vertices, E is the number of edges. So the best solution is **"Disjoint Sets":**
Disjoint sets are sets whose intersection is the empty set so it means that they don't have any element in common.

Consider following example:



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

- **https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/**
- **https://www.javatpoint.com/minimum-spanning-tree-introduction**
- **https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/**

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -7 Page No -7.47

Course Faculty

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

L25

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **III-Graphs**

Date of Lecture: 24.09.2020

Topic of Lecture: Prim's algorithm And Kruskal's algorithm.

Introduction :

- Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized
- Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which, we can traverse every vertex of the graph.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Graphs

Detailed content of the Lecture:

Prim's Algorithm

- Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

- Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

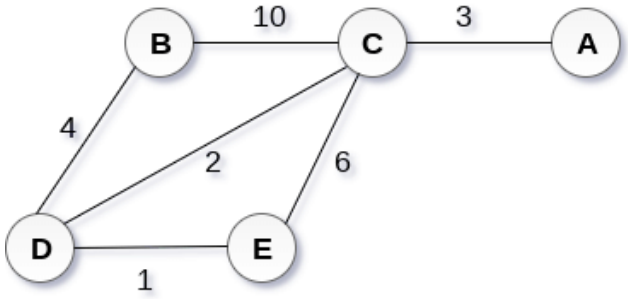
The algorithm is given as follows.

Algorithm

Step1: Select a starting vertex

- Step2: Repeat Steps 3 and 4 until there are fringe vertices

- Step 3: Select an edge e connecting the tree vertex and fringe vertex that has minimum weight
- Step 4: Add the selected edge and the vertex to the minimum spanning tree T
- [END OF LOOP]
- Step 5: EXIT
- *Example :*
- Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



- Solution
- Step 1 : Choose a starting vertex B.
- Step 2: Add the vertices that are adjacent to A. the edges that connecting the vertices are shown by dotted lines.
- Step 3: Choose the edge with the minimum weight among all. i.e. BD and add it to MST. Add the adjacent vertices of D i.e. C and E.
- Step 3: Choose the edge with the minimum weight among all. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C i.e. E and A.
- Step 4: Choose the edge with the minimum weight i.e. CA. We can't choose CE as it would cause cycle in the graph.
- The graph produces in the step 4 is the minimum spanning tree of the graph shown in the above figure.

    - The cost of MST will be calculated as
    - cost(MST) = 4 + 2 + 1 + 3 = 10 units.
        - *Kruskal's Algorithm*

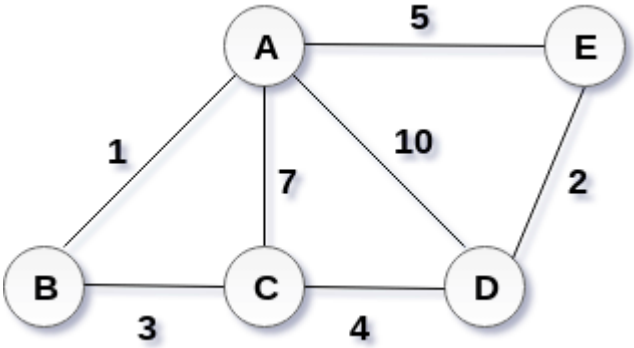
Kruskal's Algorithm

- It is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which, we can traverse every vertex of the graph. Kruskal's algorithm follows greedy approach which finds an optimum solution at every stage instead of focusing on a global optimum.

    - The Kruskal's algorithm is given as follows.

    - Algorithm
    - Step 1: Create a forest in such a way that each graph is a separate tree.
    - Step 2: Create a priority queue Q that contains all the edges of the graph.

- Step 3: Repeat Steps 4 and 5 while Q is NOT EMPTY
- Step 4: Remove an edge from Q
- Step 5: IF the edge obtained in Step 4 connects two different trees, then Add it to the forest (for combining two trees into one tree).
- ELSE
- Discard the edge
- Step 6: END
- Example :
- Apply the Kruskal's algorithm on the graph given as follows.
-

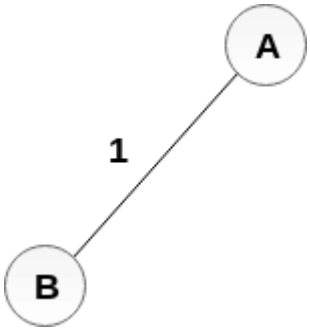


  - Edge AE AD AC AB BC CD DE
  - Weight 5 10 7 1 3 4 2

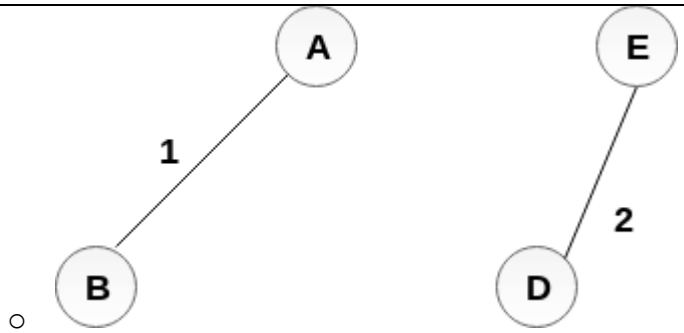
    Sort the edges according to their weights.

- Edge AB DE BC CD AE AC AD
- Weight 1 2 3 4 5 7 10
- Start constructing the tree;
- Add AB to the MST;

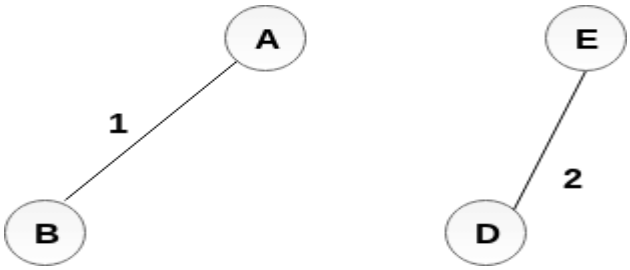
- *Kruskal's Algorithm*
- Add DE to the MST;
-



  - **Add DE to the MST;**



- **Add BC to the MST;**



- The next step is to add AE, but we can't add that as it will cause a cycle.
- The next edge to be added is AC, but it can't be added as it will cause a cycle.
- The next edge to be added is AD, but it can't be added as it will contain a cycle.
- Hence, the final MST is the one which is shown in the step 4.
- the cost of MST = 1 + 2 + 3 + 4 = 10.

- **Video Content / Details of website for further learning (if any):** **https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/ https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/**

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

- J.John Manoj Kumar ,P.Sudharsan ,"Data Structures using C" RBA Publications, Chennai" – Chapter -7 Page No -7.47

Course Faculty

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

L26

LECTURE HANDOUTS

MCA

I / I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit :**III-Graphs.**

Date of Lecture: 25.09.2020

Topic of Lecture: Biconnectivity

Introduction :

- An undirected graph is said to be a biconnected graph, if there are two vertex-disjoint paths between any two vertices are present. In other words, we can say that there is a cycle between any two vertices.

Prerequisite knowledge for Complete understanding and learning of Topic:

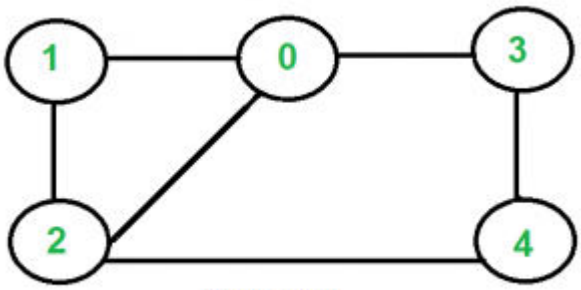
- Basics of Graph.

Detailed content of the Lecture:

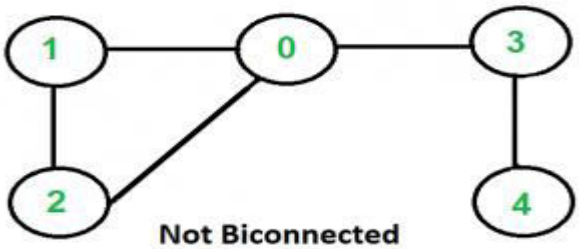
Biconnected graph:

- An undirected graph is called Biconnected if there are two vertex-disjoint paths between any two vertices. In a Biconnected Graph, there is a simple cycle through any two vertices.
- By convention, two nodes connected by an edge form a biconnected graph, but this does not verify the above properties. For a graph with more than two vertices, the above properties must be there for it to be Biconnected.
- Or in other words:
- A graph is said to be Biconnected if:
    - It is connected, i.e. it is possible to reach every vertex from every other vertex, by a simple path.
    - Even after removing any vertex the graph remains connected.

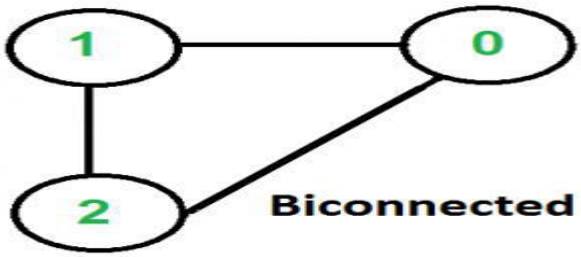
Biconnected



Biconnected



Not Biconnected



Biconnected

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

- **https://www.geeksforgeeks.org/biconnectivity-in-a-graph/**
- **https://www.tutorialspoint.com/Biconnected-Graph**

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

- Yedidyah Langsam, Moshe J.Augenstein, Aaron M.Tenenbaum,"Data Structures using C and c++" PHI Private Limited, New Delhi Page No -571

Course Faculty

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 - 27**

MCA | **I/I**

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit :**III-Graphs**

Date of Lecture: 28.09.2020

Topic of Lecture: Euler Circuits

Introduction : (Maximum 5 sentences)

- In graph theory, an Eulerian trail (or Eulerian path) is a trail in a finite graph that visits every edge exactly once (allowing for revisiting vertices). Similarly, an Eulerian circuit or Eulerian cycle is an Eulerian trail that starts and ends on the same vertex.

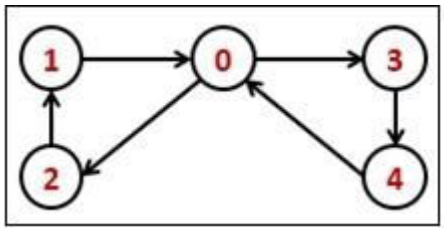
- **Prerequisite knowledge for Complete understanding and learning of Topic:**
- Basic of Graph.

Detailed content of the Lecture:

Euler Circuit in a Directed Graph:

- Eulerian path is a path in graph that visits every edge exactly once. Eulerian Circuit is an Eulerian Path which starts and ends on the same vertex.

- A graph is said to be eulerian if it has a eulerian cycle. We have discussed eulerian circuit for an undirected graph. In this post, the same is discussed for a directed graph.

- 

- To check whether a graph is Eulerian or not, we have to check two conditions −

- The graph must be connected.
- The in-degree and out-degree of each vertex must be the same.
- An Euler path is a path that uses every edge of a graph exactly once. An Euler circuit is a circuit that uses every edge of a graph exactly once. I An Euler path starts and ends at different vertices. I An Euler circuit starts and ends at the same vertex.

- Euler Paths and Euler Circuits B C E D A B C E D A An
  Euler path: BBADCDEBC
- The Criterion for Euler Paths Suppose that a graph has an Euler path P. For every vertex v other than the starting and ending vertices, the path P enters v the same number of times that it leaves v (say s times).
- The Criterion for Euler Paths Suppose that a graph has an Euler path P. For every vertex v other than the starting and ending vertices, the path P enters v the same number of times that it leaves v (say s times). Therefore, there are 2s edges having v as an endpoint.

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

- **https://www.geeksforgeeks.org/euler-circuit-directed-graph/**
- **https://www.tutorialspoint.com/Euler-Circuit-in-a-Directed-Graph**

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

- Net Reference

Course Faculty

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

LECTURE HANDOUTS

L - 28

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **IV - Introduction to Algorithms**

Date of Lecture: 01.10.2020

Topic of Lecture: Introduction to Algorithms

Introduction : (Maximum 5 sentences)

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.
- Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Types of Data
- Basics of Data Structure
- Types of Data Structure

Detailed content of the Lecture:

- From the data structure point of view, following are some important categories of algorithms
- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.

Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics

- **Unambiguous** − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** − An algorithm should have 0 or more well-defined inputs.
- **Output** − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** − Algorithms must terminate after a finite number of steps.
- **Feasibility** − Should be feasible with the available resources.

- **Independent** − An algorithm should have step-by-step directions, which should be independent of any programming code.

Advantages of Algorithms:

- It is easy to understand.
- Algorithm is a step-wise representation of a solution to a given problem.
- In Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

Disadvantages of Algorithms:

- Writing an algorithm takes a long time so it is time-consuming.
- Branching and Looping statements are difficult to show in Algorithms.

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

1. **https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm**
2. **https://www.geeksforgeeks.org/introduction-to-algorithms**
3. **https://www.youtube.com/watch?v=4RLhuZ3N9nc**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -25

Course Faculty

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

LECTURE HANDOUTS

L - 29

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **IV - Introduction to Algorithms**

Date of Lecture: 03.10.2020

Topic of Lecture: Notion of Algorithm

Introduction : (Maximum 5 sentences)

- Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a given input, n.
- There are three different notations: big O, big Theta (Θ), and big Omega (Ω).

Prerequisite knowledge for Complete understanding and learning of Topic:

- Knowledge of programming languages
- Problem-solving ability

Detailed content of the Lecture:

- An algorithm is a sequence of unambiguous instructions for solving a problem. I.e., for obtaining a required output for any legitimate input in a finite amount of time.
- There are various methods to solve the same problem.

The important points to be remembered are:

1. The non-ambiguity requirement for each step of an algorithm cannot be compromised.
2. The range of input for which an algorithm works has to be specified carefully.
3. The same algorithm can be represented in different ways.
4. Several algorithms for solving the same problem may exist.

5. Algorithms for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds.

The example here is to find the gcd of two integers with three different ways:

- The gcd of two nonnegative, not-both –zero integers m & n, denoted as gcd (m, n) is defined as the largest integer that divides both m & n evenly, i.e., with a remainder of zero.
- Euclid of Alexandria outlined an algorithm, for solving this problem in one of the volumes of his Elements.
- **Gcd (m, n) = gcd (n, m mod n)**

is applied repeatedly until m mod n is equal to 0;

since gcd (m, o) = m. {the last value of m is also the gcd of the initial m & n.}

The structured description of this algorithm is:

Step 1: If n=0, return the value of m as the answer and stop; otherwise, proceed to step2.

Step 2: Divide m by n and assign the value of the remainder to r.

Step 3: Assign the value of n to m and the value of r to n. Go to step 1.

Euclid"s algorithm is:

ALGORITHM *Euclid(m, n)*

//Computes gcd*(m, n)* by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers *m* and *n*

//Output: Greatest common divisor of *m* and *n*

while *n* _= 0 **do**

r ←*m* mod *n*

m←*n*

n←*r*

return *m*

- This algorithm comes to a stop, when the 2nd no becomes 0.
- The second number of the pair gets smaller with each iteration and it cannot become negative.
- ndeed, the new value of n on the next iteration is m mod n, which is always smaller than n.
- Hence, the value of the second number in the pair eventually becomes 0, and the algorithm stops.

Example: gcd (60,24) = gcd (24,12) = gcd (12,0) = 12.

- The second method for the same problem is: obtained from the definition itself. i.e., gcd of m & n is the largest integer that divides both numbers evenly.
- Obviously, that number cannot be greater than the second number (or) smaller of these two numbers,which we will denote by t = min {m, n }.
- So start checking whether t divides both m and n: if it does t is the answer ; if it doesn't t is

decreased by 1 and try again. (Do this repeatedly till you reach 12 and then stop for the example given below)

Consecutive integer checking algorithm:

Step 1: Assign the value of min {m,n} to t.

Step 2: Divide m by t. If the remainder of this division is 0, go to step 3; otherwise go to step 4.

Step 3: Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise, proceed to step 4.

Step 4: Decrease the value of t by 1. Go to step 2.

Note : this algorithm, will not work when one of its input is zero. So we have to specify the range of input explicitly and carefully. http://www.francisxavier.ac.in

The third procedure is as follows:

Step 1: Find the prime factors of m.

Step 2: Find the prime factors of n.

Step 3: Identify all the common factors in the two prime expansions found in step 1 & 2. (If p is a common factor occurring pm & pn times is m and n, respectively,it should be repeated min { pm, pn } times.).

Step 4: Compute the product of all the common factors and return it as gcd of the numbers given.

Example: 60 = 2.2.3.5 24 = 2.2.2.3

gcd (60,24) = 2.2.3 = 12 .

- This procedure is more complex and ambiguity arises since the prime factorization is not defined. So to make it as an efficient algorithm, incorporate the algorithm to find the prime factors.

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

1. https://www.geeksforgeeks.org/introduction-to-algorithms/

2. https://www.youtube.com/watch?v=DBFZBWzNuEc

3. https://www.youtube.com/watch?v=4RLhuZ3N9nc

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -29

Course Faculty

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 - 30

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **IV - Introduction to Algorithms**

Date of Lecture: 07.10.2020

Topic of Lecture: Fundamentals of Algorithmic problem solving

- An algorithm is a set of rules that **specify the order and kind of arithmetic operations** that are used on a specified set of data.
- An algorithm is an effective method for solving a problem using a finite sequence of instructions. An algorithm is a finite, definite, effective procedure, with some output.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Problem-solving ability
- Types of Data Structure
- Introduction of Algorithms

Detailed content of the Lecture:

- Algorithms can be considered to be procedural solutions to problems. There are certain steps to be followed in designing and analyzing an algorithm.
- Code the algorithm

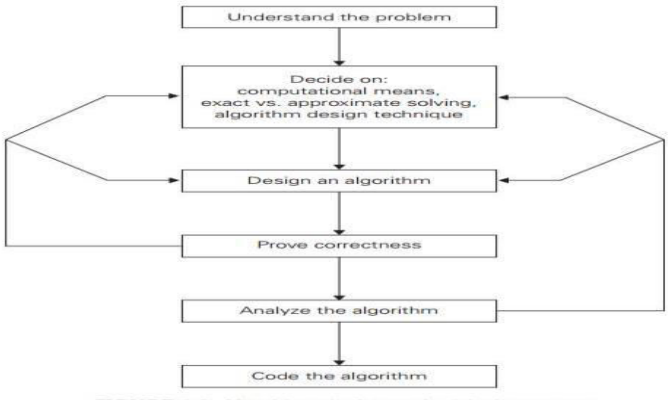


FIGURE 1.2 Algorithm design and analysis process.

Understanding the problem

- An input to an algorithm specifies an instance of the problem the algorithm solves.

- It's also important to specify exactly the range of instances the algorithm needs to handle.
- Before this we have to clearly understand the problem and clarify the doubts after leading the problems description.
- Correct algorithm should work for all possible inputs.

Ascertaining the capabilities of a computational Device

- The second step is to ascertain the capabilities of a machine.
- The essence of von-Neumann machines architecture is captured by RAM, Here the instructions are executed one after another, one operation at a time, Algorithms designed to be executed on such machines are called sequential algorithms.

Choosing between exact and approximate problem solving

- The next decision is to choose between solving the problem exactly or solving it approximately.
- Based on this, the algorithms are classified as exact and approximation algorithms.
- There are three issues to choose an approximation algorithm.
- First, there are certain problems like extracting square roots, solving non-linear equations which cannot be solved exactly.
- Secondly, if the problem is complicated it slows the operations. **E.g. traveling salesman problem**.
- Third, this algorithm can be a part of a more sophisticated algorithm that solves a problem exactly.

Deciding on data structures

- Data structures play a vital role in designing and analyzing the algorithms.
- Some of the algorithm design techniques also depend on the structuring data specifying a problem's instance.
- **Algorithm + Data structure = Programs**

Algorithm Design Techniques

- An algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- Learning these techniques are important for two reasons, First, they provide guidance for designing for new problems.
- Second, algorithms are the cornerstones of computer science.
- Algorithm design techniques make it possible to classify algorithms according to an underlying design idea; therefore, they can serve as a natural way to both categorize and study algorithms.

Methods of specifying an Algorithm

- A psuedocode , which is a mixture of a natural language and programming language like constructs.
- Its usage is similar to algorithm descriptions for writing psuedocode there are some dialects which omits declarations of variables, use indentation to show the scope of the statements such as if, for and while.
- **Use → for assignment operations, (//) two slashes for comments.**
- To specify algorithm flowchart is used which is a method of expressing an algorithm by a collection of connected geometric shapes consisting descriptions of the algorithm's steps.

Proving an Algorithm's correctness

- Correctness has to be proved for every algorithm.
- To prove that the algorithm gives the required result for every legitimate input in a finite amount

of time. For some algorithms, a proof of correctness is quite easy; for others it can be quite complex.

- A technique used for proving correctness s by mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.
- But we need one instance of its input for which the algorithm fails.
- If it is incorrect, redesign the algorithm, with the same decisions of data structures design technique etc.
- The notion of correctness for approximation algorithms is less straightforward than it is for exact algorithm.
- **For example,** in gcd (m,n) two observations are made. One is the second number gets smaller on every iteration and the algorithm stops when the second number becomes 0.

Analyzing an algorithm

- There are two kinds of algorithm efficiency: time and space efficiency.
- Time efficiency indicates how fast the algorithm runs; space efficiency indicates how much extra memory the algorithm needs.
- Another desirable characteristic is simplicity.
- Simper algorithms are easier to understand and program, the resulting programs will be easier to debug.
- For e.g. Euclid's algorithm to fid gcd (m,n) is simple than the algorithm which uses the prime factorization.
- Another desirable characteristic is generality. Two issues here are generality of the problem the algorithm solves and the range of inputs it accepts.
- The designing of algorithm in general terms is sometimes easier.
- For eg, the general problem of computing the gcd of two integers and to solve the problem.

Coding an algorithm

- Programming the algorithm by using some programming language.
- Formal verification is done for small programs.
- Validity is done thru testing and debugging.
- Inputs should fall within a range and hence require no verification.
- Some compilers allow code optimization which can speed up a program by a constant factor whereas a better algorithm can make a difference in their running time.
- The analysis has to be done in various sets of inputs.
- A good algorithm is a result of repeated effort & work.
- The program's stopping / terminating condition has to be set.

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

1. **https://www.brainkart.com/article/Fundamentals-of-Algorithmic-Problem-Solving_7992**
2. **https://www.geeksforgeeks.org/how-to-use-algorithms-to-solve-problems/**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -33

Course Faculty

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

LECTURE HANDOUTS

L - 31

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **IV - Introduction to Algorithms**

Date of Lecture: 09.10.2020

Topic of Lecture: Important problem types

Introduction : (Maximum 5 sentences)

- An Algorithm is a sequence of steps to solve a problem.
- Design and Analysis of Algorithm is very important for designing algorithm to **solve different types of problems** in the branch of computer science and information technology.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Types of Data
- Basics of Data Structure
- Types of Data Structure

Detailed content of the Lecture:

- The two motivating forces for any problem is its practical importance and some specific characteristics.

The different types are:

- Sorting
- Searching
- String processing
- Graph
- Combinatorial problems
- Geometric problems
- Numerical problems.

1. Sorting

- Sorting problem is one which rearranges the items of a given list in ascending order.
- We usually sort a list of numbers, characters, strings and records similar to college information about their students, library information and company information is chosen for guiding the sorting technique.
- For eg in student's information, we can sort it either based on student's register number or by

their names. Such pieces of information is called a **key.**

- There are two important properties. The first is called stable, if it preserves the relative order of any two equal elements in its input.
- For example, if we sort the student list based on their GPA and if two students GPA are the same, then the elements are stored or sorted based on its position.
- The second is said to be 'in place' if it does not require extra memory. There are some sorting algorithms that are in place and those that are not.

2. Searching

- The searching problem deals with finding a given value, called a search key, in a given set. The searching can be either a straightforward algorithm or binary search algorithm which is a different form.
- These algorithms play a important role in real-life applications because they are used for storing and retrieving information from large databases.
- Some algorithms work faster but require more memory, some are very fast but applicable only to sorted arrays. Searching, mainly deals with addition and deletion of records.
- In such cases, the data structures and algorithms are chosen to balance among the required set of operations.

3.String processing

- A String is a sequence of characters. It is mainly used in string handling algorithms. Most common ones are text strings, which consists of letters, numbers and special characters.
- Bit strings consist of zeroes and ones. The most important problem is the string matching, which is used for searching a given word in a text.
- For e.g. **sequential searching and brute- force string matching algorithms.**

4.Graph problems

- One of the interesting area in algorithmic is graph algorithms.
- A graph is a collection of points called vertices which are connected by line segments called edges. Graphs are used for modeling a wide variety of real-life applications such as transportation and communication networks.
- It includes graph traversal, shortest-path and topological sorting algorithms.
- Some graph problems are very hard, only very small instances of the problems can be solved in realistic amount of time even with fastest computers.
- There are two common problems: the traveling salesman problem, finding the shortest tour through n cities that visits every city exactly once
- The **graph-coloring problem** is to assign the smallest number of colors to vertices of a graph

so that no two adjacent vertices are of the same color.

5.Combinatorial problems

- The traveling salesman problem and the graph-coloring problem are examples of combinatorial problems.
- These are problems that ask us to find a combinatorial object such as permutation, combination or a subset that satisfies certain constraints and has some desired (e.g. maximizes a value or minimizes a cost).
- These problems are difficult to solve for the following facts.
- First, the number of combinatorial objects grows extremely fast with a problem's size.
- Second, there are no known algorithms, which are solved in acceptable amount of time.

6. **Geometric problems**

- Geometric algorithms deal with geometric objects such as points, lines and polygons.
- It also includes various geometric shapes such as triangles, circles etc.
- The applications for these algorithms are in computer graphic, robotics etc.The two problems most widely used are the closest-pair problem, given 'n' points in the plane, find the closest pair among them.
- **The convex-hull problem** is to find the smallest convex polygon that would include all the points of a given set.

7.Numerical problems

- This is another large special area of applications, where the problems involve mathematical objects of continuous nature: solving equations computing definite integrals and evaluating functions and so on. These problems can be solved only approximately.
- These require real numbers, which can be represented in a computer only approximately.
- If can also lead to an accumulation of round-off errors.
- The algorithms designed are mainly used in scientific and engineering applications

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

1. **https://www.brainkart.com/article/Important-Problem-Types-in-Algorithms-Analysis**
2. **https://www.srividyaengg.ac.in/coursematerial/CSE/104441.pdf**
3. **https://www.youtube.com/watch?v=V7-Nw8B7MAI**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -43

Course Faculty

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

LECTURE HANDOUTS

L - 32

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **IV - Introduction to Algorithms**

Date of Lecture: 12.10.2020

Topic of Lecture: Mathematical analysis for recursive algorithms

Introduction : (Maximum 5 sentences)

- Design a recursive algorithm for computing 2n for any nonnegative integer n that is based on the formula **2n = 2n−1 + 2n−1**.
- Set up a recurrence relation for the number of additions made by the algorithm and solve it.
- Draw a tree of recursive calls for this algorithm and count the number of calls made by the algorithm.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Introduction of Algorithms
- Basics of Recursive Algorithms

Detailed content

- In this section, we systematically apply the general framework to analyze the efficiency of recursive algorithms.
- Let us start with a very simple example that demonstrates all the principal steps typically taken in analyzing recursive algorithms.

Example 1: **Compute the factorial function F(n) = n! for an arbitrary non negative integer n. Since, n! = 1 * 2 * ……. * (n-1) *n = n(n-1)! For n ≥ 1 and 0! = 1 by definition, we can compute F(n) = F(n-1).n with the following recursive algorithm.**

ALGORITHM F(n)

// Computes n! recursively

// Input: A nonnegative integer n

// Output: The value of n!

ifn =0 return 1

else return F(n — 1) * n

- For simplicity, we consider n itself as an indicator of this algorithm's input size (rather than the number of bits in its binary expansion).
- The basic operation of the algorithm is multiplication, whose number of executions we denote

M(n). Since the function F(n) is computed according to the formula

- **F(n) = F (n - 1) - n for n > 0,** the number of multiplications M(n) needed to compute it must satisfy the equality

M(n) = M(n-1) + 1 for n > 0 1 for n = 0

To Compute to Multiply

F(n-1) F(n-1) by n

- Indeed ,M(n-1) multiplications are spent to compute F(n - 1), and one more multiplication is needed to multiply the result by n.
- The last equation defines the sequence M(n) that we need to find.
- Note that the equation defines M(n) not explicitly, i.e., as a function of n, but implicitly as a function of its value at another point, namely n — 1.
- Such equations are called recurrence relations or, for brevity, recurrences. Recurrence relations play an important role not only in analysis of algorithms but also in some areas of applied mathematics.
- Our goal now is to solve the recurrence relation M(n) = M(n — 1) + 1, i.e., to find an explicit formula for the sequence M(n) in terms of n only.
- Note, however, that there is not one but infinitely many sequences that satisfy this recurrence.
- To determine a solution uniquely, we need an initial condition that tells us the value with which the sequence starts.
- We can obtain this value by inspecting the condition that makes the algorithm stop its recursive calls:

if n = 0 return 1.

- Generalizing our experience with investigating the recursive algorithm for computing n!, we can now outline a general plan for investigating recursive algorithms.

A General Plan for Analyzing Efficiency of Recursive Algorithms

1. Decide on a parameter (or parameters) indicating an input's size.
2. Identify the algorithm's basic operation.
3. Check whether the number of times the basic operation is executed can vary on different inputs of the same size; if it can, the worst-case, average-case, and best-case efficiencies must be investigated separately.

4. Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed.

5. Solve the recurrence or at least ascertain the order of growth of its solution.

ALGORITHM *F(n)*
//Computes the nth Fibonacci number recursively by using its definition //Input: A nonnegative integer *n*
//Output: The nth Fibonacci number
if *n* < 1 return *n*
else return *F*(*n* - 1) + *F*(*n* - 2)

Analysis:

- The algorithm's basic operation is clearly addition, so let A(n) be the number of additions per
  numbers of additions needed for computing *F(n* — 1) and *F(n* — 2) are *A(n* — 1) and *A(n* —
  to compute their sum. Thus, we get the following recurrence for
- A(n):

A(n) = A(n - 1) + *A(n* - 2) + 1 for n > 1,

A(0)=0, A(1) = 0.

The recurrence A(n) — A(n — 1) — A(n — 2) = 1 is quite similar to recurrence (2.7) but its right-ha
side is not equal to zero. Such recurrences are called ***inhomo-geneous recurrences.***

ALGORITHM *Fib(n)*

- //Computes the nth Fibonacci number iteratively by using its definition
- //Input: A nonnegative integer *n* //Output: The nth Fibonacci number F[0]<-0; F[1]<-1
- for i <- 2 to *n* do
- F[i]«-F[i-1]+F[i-2] **return** *F[n]*
- This algorithm clearly makes *n* - 1 additions. Hence, it is linear as a function of *n* and "only"
  exponential as a function of the number of bits *b* in n's binary representation.

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

1. **https://www.brainkart.com/article/Mathematical-Analysis-of-Recursive-Algorithms_8005/**
2. **https://www.youtube.com/watch?v=CbB1zivs4EY**
3. **https://www.rgpvonline.com/answer/analysis-and-design-of-algorithms/2.html**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -93

Course Faculty

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 - 33**

Co | **MCA** | **Code** : 19CAB02 - Data Structures and Algorithms | **I/I**

Course Faculty : Mrs.G.Krishnaveni

Unit : IV - Introduction to Algorithms

Date of Lecture: 19.10.2020

Topic of Lecture: Mathematical analysis for non recursive algorithms

Introduction : (Maximum 5 sentences)

- The general framework outlined is applied to analyze the efficiency of nonrecursive algorithms.
- Let us start with a very simple example that demonstrates all the principal steps typically taken in analyzing such algorithms.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Introduction of Algorithms
- Basics of non-recursive Algorithms

Detailed content of the Lecture:

EXAMPLE 1 Consider the problem of finding the value of the largest element in a list of n numbers. For simplicity, we assume that the list is implemented as an array. The following is a pseudocode of a standard algorithm for solving the problem.

ALGORITHM MaxElement(A[0,..n - 1])

//Determines the value of the largest element in a given array //Input: An array A[0..n - 1] of real numbers

//Output: The value of the largest element in A maxval <- A[0]

for i <- 1 to n - 1 do **if** A[i] > maxval maxval <— A[i]

return maxval

- The obvious measure of an input's size here is the number of elements in the array, i.e., n.
- The operations that are going to be executed most often are in the algorithm's for loop.
- There are two operations in the loop's body: the comparison **A[i] > maxval** and the assignment **maxval <- A[i].**
- Since the comparison is executed on each repetition of the loop and the assignment is not, we should consider the comparison to be the algorithm's basic operation.

- (Note that the number of comparisons will be the same for all arrays of size n; therefore, in terms of this metric, there is no need to distinguish among the worst, average, and best cases here.) Let us denote C(n) the number of times this comparison is executed and try to find a formula expressing it as a function of size n.
- The algorithm makes one comparison on each execution of the loop, which is repeated for each value of the loop's variable i within the bounds between 1 and n — 1 (inclusively).

Therefore, we get the following sum for C(n): n-1

C(n) 1

= Σ

i=1

- This is an easy sum to compute because it is nothing else but 1 repeated n — 1 times.
- Thus, http://www.francisxavier.ac.in

n-1

C(n) = Σ 1 = n-1 Є θ(n)

i=1

- Here is a general plan to follow in analyzing nonrecursive algorithms.

General Plan for Analyzing Efficiency of Nonrecursive Algorithms

1. Decide on a parameter (or parameters) indicating an input's size.

2. Identify the algorithm's basic operation. (As a rule, it is located in its innermost loop.)

3. Check whether the number of times the basic operation is executed depends only on the size of an input. If it also depends on some additional property, the worst-case, average-case, and, if necessary, best-case efficiencies have to be investigated separately. 4. Set up a sum expressing the number of times the algorithm's basic operation is executed.

5. Using standard formulas and rules of sum manipulation either find a closed-form formula for the count or, at the very least, establish its order of growth.

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

1. https://www.rgpvonline.com/answer/analysis-and-design-of-algorithms/2.html
2. https://www.brainkart.com/article/Mathematical-Analysis-of-Recursive-Algorithms_8005/
3. https://www.youtube.com/watch?v=2fYmmHLJs5M

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -85

Course Faculty

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

LECTURE HANDOUTS

L - 34

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **IV - Introduction to Algorithms**

Date of Lecture: 26.10.2020

Topic of Lecture: Brute Force

- Brute force is a straightforward approach to solving a problem, usually directly based on the problem's statement and definitions of the concepts involved.
- For e.g. the algorithm to find the gcd of two numbers.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Introduction of Algorithms
- Basics of non-recursive Algorithms

Detailed content of the Lecture:

Brute force approach is not an important algorithm design strategy for the following reasons:

- First, unlike some of the other strategies, brute force is applicable to a very wide variety of problems. Its used for many elementary but algorithmic tasks such as computing the sum of n numbers, finding the largest element in a list and so on.
- Second, for some problem it yields reasonable algorithms of at least some practical value with no limitation on instance size.
- Third, the expense of designing a more efficient algorithm if few instances to be solved and with acceptable speed for solving it.
- Fourth, even though it is inefficient, it can be used to solve small-instances of a problem.
- Last, it can serve as an important theoretical or educational propose.

2.1.1 CLOSEST PAIR AND CONVEX HULL PROBLEMS

CLOSEST-PAIR PROBLEM

- The closest-pair problem calls for finding the two closest points in a set of *n* points.
- It is the simplest of a variety of problems in computational geometry that deals with proximity of points in the plane or higher-dimensional spaces.

- Points in question can represent such physical objects as airplanes or post offices as well as database records, statistical samples, DNA sequences, and so on. An air-traffic controller might be interested in two closest planes as the most probable collision candidates.
- Aregional postal service manager might need a solution to the closestpair problem to find candidate post-office locations to be closed.
- One of the important applications of the closest-pair problem is cluster analysis in statistics. Based on *n* data points, hierarchical cluster analysis seeks to organize them in a hierarchy of clusters based on some similarity metric.
- For numerical data, this metric is usually the Euclidean distance; for

http://www.francisxavier.ac.in

- For text and other nonnumerical data, metrics such as the Hamming distance are used. A bottom-up algorithm begins with each element as a separate cluster and merges them into successively larger clusters by combining the closest pair of clusters.

- For simplicity, we consider the two-dimensional case of the closest-pair problem. We assume that the points in question are specified in a standard fashion by their *(x, y)* Cartesian coordinates and that the distance between two points *pi(xi,yi)* and *pj(xj, yj)* is the standard Euclidean distance

$$d(pi, pj) = (xi - xj)2 + (yi - yj)2.$$

- The brute-force approach to solving this problem leads to the following obvious algorithm: compute the distance between each pair of distinct points and find a pair with the smallest distance. Of course, we do not want to compute the distance between the same pair of points twice.
- To avoid doing so, we consider only the pairs of points *(pi, pj)* for which $i < j$.

Pseudocode below computes the distance between the two closest points; getting the closest points themselves requires just a trivial modification.

ALGORITHM *BruteForceClosestPair(P)*

//Finds distance between two closest points in the plane by brute force

//Input: A list *P* of n $(n \geq 2)$ points $p1(x1, y1), \ldots, pn(xn, yn)$

//Output: The distance between the closest pair of points

$d \leftarrow \infty$

for $i \leftarrow 1$ **to** $n - 1$ **do**

for $j \leftarrow i + 1$ **to** n **do**

$d \leftarrow \min(d, sqrt((xi$ $- xj)2 + (yi$ $- yj)2))$ //*sqrt* is square root **return** *d*
Video Content / Details of website for further learning (if any): 1. https://www.freecodecamp.org/news/brute-force-algorithms-explained/ 2. https://www.geeksforgeeks.org/brute-force-approach-and-its-pros-and-cons/
Important Books/Journals for further learning including the page nos.: • Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -121

Course Faculty

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

LECTURE HANDOUTS

L - 35

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **IV - Introduction to Algorithms**

Date of Lecture: 27.10.2020

Topic of Lecture: Selection Sort

Introduction : (Maximum 5 sentences)

- Selection sort is a simple sorting algorithm.
- This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end.
- Initially, the sorted part is empty and the unsorted part is the entire list.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Algorithm
- Introduction of Sorting

Detailed content of the Lecture:

- The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

  1) The subarray which is already sorted.

  2) Remaining subarray which is unsorted.

  In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

- Following example explains the above steps:

  arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]

// and place it at beginning

11 25 12 22 64

// Find the minimum element in arr[1...4]

// and place it at beginning of arr[1...4]

11 **12** 25 22 64

// Find the minimum element in arr[2...4]

// and place it at beginning of arr[2...4]

11 12 **22** 25 64

// Find the minimum element in arr[3...4]

// and place it at beginning of arr[3...4]

11 12 22 **25** 64

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

1. **https://www.geeksforgeeks.org/selection-sort/**
2. **https://www.tutorialspoint.com/data_structures_algorithms/selection_sort_algorithm.htm**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -123

Course Faculty

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

LECTURE HANDOUTS

L - 36

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **IV - Introduction to Algorithms**

Date of Lecture: 28.10.2020

Topic of Lecture: Bubble Sort

Introduction : (Maximum 5 sentences)

- Bubble Sort, also known as Exchange Sort, is a simple sorting algorithm.
- It works by repeatedly stepping throughout the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Introduction of Algorithms
- Basics of Sorting

Detailed content of the Lecture:

- Bubble Sort, also known as Exchange Sort, is a simple sorting algorithm. It works by repeatedly stepping throughout the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order.
- Bubble Sort is an elementary sorting algorithm, which works by repeatedly exchanging adjacent elements, if necessary. When no exchanges are required, the file is sorted.
- This is the simplest technique among all sorting algorithms.

Algorithm: Sequential-Bubble-Sort (A)

```
fori← 1 to length [A] do
for j ← length [A] down-to i +1 do
  if A[A] < A[j - 1] then
    Exchange A[j] ↔ A[j-1]
```

Memory Requirement

From the algorithm stated above, it is clear that bubble sort does not require extra memory.

Example

Unsorted list:	5	2	1	4	3	7	6

1st iteration:

5 > 2 swap	2	5	1	4	3	7	6
5 > 1 swap	2	1	5	4	3	7	6
5 > 4 swap	2	1	4	5	3	7	6
5 > 3 swap	2	1	4	3	5	7	6
5 < 7 no swap	2	1	4	3	5	7	6
7 > 6 swap	2	1	4	3	5	6	7

2nd iteration:

2 > 1 swap	1	2	4	3	5	6	7
2 < 4 no swap	1	2	4	3	5	6	7
4 > 3 swap	1	2	3	4	5	6	7
4 < 5 no swap	1	2	3	4	5	6	7
5 < 6 no swap	1	2	3	4	5	6	7

There is no change in 3rd, 4th, 5th and 6th iteration.

Finally,

the sorted list is	1	2	3	4	5	6	7

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

1. **https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_bubble_sort.htm**
2. **https://www.javatpoint.com/daa-bubble-sort**
3. **https://www.geeksforgeeks.org/bubble-sort/**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -124

Course Faculty

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

L - 37

LECTURE HANDOUTS

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 02.11.2020

Topic of Lecture: Algorithm Analysis

Introduction : (Maximum 5 sentences)

- Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Most algorithms are designed to work with inputs of arbitrary length.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Algorithm
- Design of Algorithms

Detailed content of the Lecture:

Need for Analysis

Algorithms are often quite different from one another, though the objective of these algorithms are the same.

- For example, we know that a set of numbers can be sorted using different algorithms. Number of comparisons performed by one algorithm may vary with others for the same input.

Generally, we perform the following types of analysis −

- **Worst-case** − The maximum number of steps taken on any instance of size **a**.
- **Best-case** − The minimum number of steps taken on any instance of size **a**.
- **Average case** − An average number of steps taken on any instance of size **a**.
- **Amortized** − A sequence of operations applied to the input of size **a** averaged over time.
- In this context, if we compare **bubble sort** and **merge sort**.
- Bubble sort does not require additional memory, but merge sort requires additional space.
- Though time complexity of bubble sort is higher compared to merge sort, we may need to apply bubble sort if the program needs to run in an environment, where memory is very limited.
- To solve a problem, we need to consider time as well as space complexity as the program may run on a system where memory is limited but adequate space is available or may be vice-versa.

Asymptotic Analysis

- The asymptotic behavior of a function ***f(n)*** refers to the growth of ***f(n)*** as **n** gets large.
- We typically ignore small values of **n**, since we are usually interested in estimating how slow the program will be on large inputs.
- A good rule of thumb is that the slower the asymptotic growth rate, the better the algorithm.
- A recurrence relation can be solved using the following methods −
- **Substitution Method** − In this method, we guess a bound and using mathematical induction we prove that our assumption was correct.
- **Recursion Tree Method** − In this method, a recurrence tree is formed where each node represents the cost.
- **Master's Theorem** − This is another important technique to find the complexity of a recurrence relation.

Amortized Analysis

- Amortized analysis is generally used for certain algorithms where a sequence of similar operations are performed.
- Amortized analysis provides a bound on the actual cost of the entire sequence, instead of bounding the cost of sequence of operations separately.
- The complexity of an algorithm describes the efficiency of the algorithm in terms of the amount of the memory required to process the data and the processing time.

Complexity of an algorithm is analyzed in two perspectives: **Time** and **Space**.

Time Complexity

- It's a function describing the amount of time required to run an algorithm in terms of the size of the input.
- "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take.

Space Complexity

- It's a function describing the amount of memory an algorithm takes in terms of the size of input to the algorithm.
- We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this.
- Space complexity is sometimes ignored because the space used is minimal and/or obvious, however sometimes it becomes as important an issue as time.

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

- **https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_asymptotic_notations_apriori.htm**
- **https://www.javatpoint.com/daa-tutorial**
-

- **https://www.geeksforgeeks.org/analysis-of-algorithms-set-1-asymptotic-analysis/**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -66

Course Faculty

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 - 38

LECTURE HANDOUTS

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 05.11.2020

Topic of Lecture: Asymptotic Notations

Introduction : (Maximum 5 sentences)

- Resources for an algorithm are usually expressed as a function regarding input. Often this function is messy and complicated to work.
- To study Function growth efficiently, we reduce the function down to the important part.
- Let f (n) = an2+bn+c

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Algorithm
- Design of Algorithms

Detailed content of the Lecture:

- The word **Asymptotic** means approaching a value or curve arbitrarily closely (i.e., as some sort of limit is taken).
- It is a technique of representing limiting behavior. The methodology has the applications across science.
- It can be used to analyze the performance of an algorithm for some large data set.
- In computer science in the analysis of algorithms, considering the performance of algorithms when applied to very large input datasets.
- The simplest example is a function $f(n) = n^2+3n$, the term 3n becomes insignificant compared to n^2 when n is very large. The function "$f(n)$ is said to be **asymptotically equivalent** to n^2 as $n \to \infty$", and here is written symbolically as $f(n) \sim n^2$.

Asymptotic notations

Notations are used to write fastest and slowest possible running time for an algorithm.

- These are also referred to as 'best case' and 'worst case' scenarios respectively.
- "In asymptotic notations, we derive the complexity concerning the size of the input. (Example in terms of n)"
- "These notations are important because without expanding the cost of running the algorithm, we can estimate the complexity of the algorithms."
- Different types of asymptotic notations are used to represent the complexity of an algorithm.

Following asymptotic notations are used to calculate the running time complexity of an algorithm.

- O − Big Oh
- Ω − Big omega
- θ − Big theta
- o − Little Oh
- ω − Little omega
- Asymptotic Notation is a way of comparing function that ignores constant factors and small input sizes.
- Three notations are used to calculate the running time complexity of an algorithm:

Big-oh notation: Big-oh is the formal method of expressing the upper bound of an algorithm's running time. It is the measure of the longest amount of time. The function **f (n) = O (g (n))** [read as "f of n is big-oh of g of n"] if and only if exist positive constant c and such that

f (n) ⩽ k.g (n)f(n)⩽k.g(n) for n>n0n>n0 in all case

Hence, function g (n) is an upper bound for function f (n), as g (n) grows faster than f (n)

Omega () Notation: The function f (n) = Ω (g (n)) [read as "f of n is omega of g of n"] if and only if there exists positive constant c and n_0 such that

F (n) ≥ k* g (n) for all n, n≥ n_0

Hence, the complexity of **f (n)** can be represented as Ω (g (n))

Theta (θ): The function f (n) = θ (g (n)) [read as "f is the theta of g of n"] if and only if there exists positive constant k_1, k_2 and k_0 such that

k_1 * g (n) ≤ f(n)≤ k_2 g(n)for all n, n≥ n_0

Hence, the complexity of f (n) can be represented as θ (g(n)).

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

- **https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_asymptotic_notations_apriori.htm**
- **https://www.geeksforgeeks.org/analysis-of-algorithms-set-1-asymptotic-analysis/**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -76

Course Faculty

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

L - 39

LECTURE HANDOUTS

MCA | **I/I**

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 09.11.2020

Topic of Lecture: Divide and Conquer - Merge Sort

Introduction : (Maximum 5 sentences)

- **Divide and conquer approach**, a problem is divided into smaller problems, then the smaller problems are solved independently, and finally the solutions of smaller problems are combined into a solution for the large problem.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Algorithm
- Analysis of Algorithm

Detailed content of the Lecture:

Generally, divide-and-conquer algorithms have three parts −

- **Divide the problem** into a number of sub-problems that are smaller instances of the same problem.
- **Conquer the sub-problems** by solving them recursively. If they are small enough, solve the sub-problems as base cases.
- **Combine the solutions** to the sub-problems into the solution for the original problem.

Pros and cons of Divide and Conquer Approach

- Divide and conquer approach supports parallelism as sub-problems are independent.
- Hence, an algorithm, which is designed using this technique, can run on the multiprocessor system or in different machines simultaneously.
- In this approach, most of the algorithms are designed using recursion, hence memory management is very high. For recursive function stack is used, where function state needs to be stored.

Application of Divide and Conquer Approach

Following are some problems, which are solved using divide and conquer approach.

- Finding the maximum and minimum of a sequence of numbers
- Strassen's matrix multiplication
- Merge sort
- Binary search

Problem Statement

- The problem of sorting a list of numbers lends itself immediately to a divide-and-conquer strategy: split the list into two halves, recursively sort each half, and then merge the two sorted sub-lists.

Solution

In this algorithm, the numbers are stored in an array ***numbers[]***. Here, ***p*** and ***q*** represents the start and end index of a sub-array.

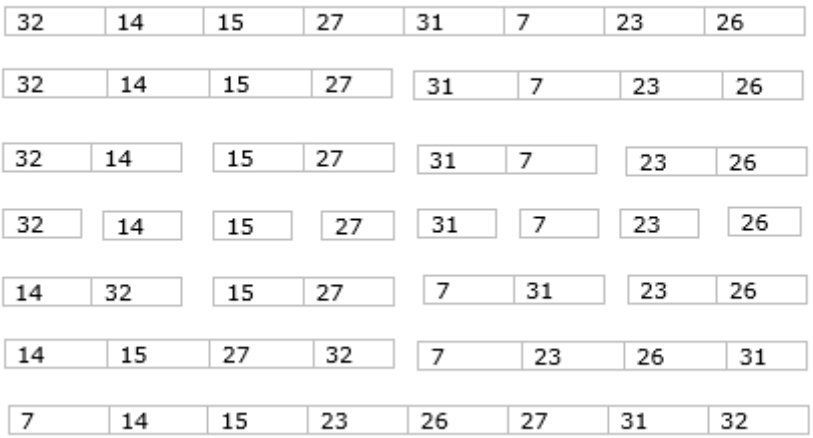
Algorithm: Merge-Sort (numbers[], p, r)

```
if p < r then
q = ⌊(p + r) / 2⌋
Merge-Sort (numbers[], p, q)
    Merge-Sort (numbers[], q + 1, r)
    Merge (numbers[], p, q, r)
Function: Merge (numbers[], p, q, r)
n1 = q – p + 1
n2 = r – q
declare leftnums[1…n1 + 1] and rightnums[1…n2 + 1] temporary arrays
for i = 1 to n1
   leftnums[i] = numbers[p + i - 1]
for j = 1 to n2
   rightnums[j] = numbers[q+ j]
leftnums[n1 + 1] = ∞
rightnums[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
   if leftnums[i] ≤ rightnums[j]
      numbers[k] = leftnums[i]
      i = i + 1
   else
      numbers[k] = rightnums[j]
      j = j + 1
```

Example

- In the following example, we have shown Merge-Sort algorithm step by step.
- First, every iteration array is divided into two sub-arrays, until the sub-array contains only one element.

Divide and Merge operations step by step:



- When these sub-arrays cannot be divided further, then merge operations are performed.

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

- **https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_merge_sort.htm**
- **https://www.geeksforgeeks.org/merge-sort**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -145

Course Faculty

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

L - 40

LECTURE HANDOUTS

MCA | **I/I**

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 12.11.2020

Topic of Lecture: Binary Search

Introduction : (Maximum 5 sentences)

- Binary Search and Ternary Search Algorithms are based **on Decrease and Conquer technique**.
- Because, you do not divide the problem, you actually decrease the problem by dividing by 2(3 in ternary search).

Prerequisite knowledge for Complete understanding and learning of Topic:

- **Basics of Algorithm**
- **Concept of Divide and Conquer**

Detailed content of the Lecture:

Problem Statement

- Binary search can be performed on a sorted array.
- In this approach, the index of an element **x** is determined if the element belongs to the list of elements. If the array is unsorted, linear search is used to determine the position.

Solution

- In this algorithm, we want to find whether element **x** belongs to a set of numbers stored in an array ***numbers[]***.
- Where ***l*** and ***r*** represent the left and right index of a sub-array in which searching operation should be performed.

Algorithm: Binary-Search(numbers[], x, l, r)

```
if l = r then
  return l
else
  m := ⌊(l + r) / 2⌋
  if x ≤ numbers[m]  then
    return Binary-Search(numbers[], x, l, m)
  else
    return Binary-Search(numbers[], x, m+1, r)
```

Analysis

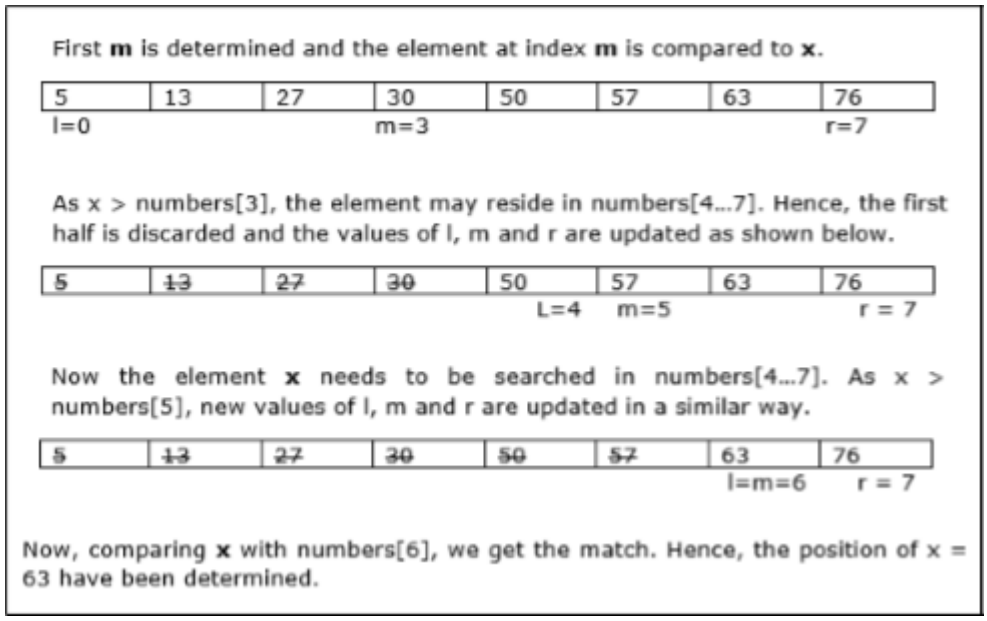
- Linear search runs in ***O(n)*** time. Whereas binary search produces the result in ***O(log n)*** time
- Let **T(n)** be the number of comparisons in worst-case in an array of **n** elements.

Hence,

T(n)={0T(n2)+1ifn=1otherwiseT(n)={0ifn=1T(n2)+1otherwise
Using this recurrence relation T(n)=lognT(n)=logn.
Therefore, binary search uses O(logn)O(logn) time.

Example

In this example, we are going to search element 63.



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

- **https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_binary_search.htm**
- **https://www.geeksforgeeks.org/divide-and-conquer/**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -157

Course Faculty

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

L - 41

LECTURE HANDOUTS

MCA | **I/I**

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 23.11.2020

Topic of Lecture: Greedy Algorithms - Knapsack Problem

Introduction : (Maximum 5 sentences)

- Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit.
- So the problems where choosing locally optimal also leads to global solution are best fit for Greedy.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Greedy Algorithm
- Concept of Knapsack Problem

Detailed content of the Lecture:

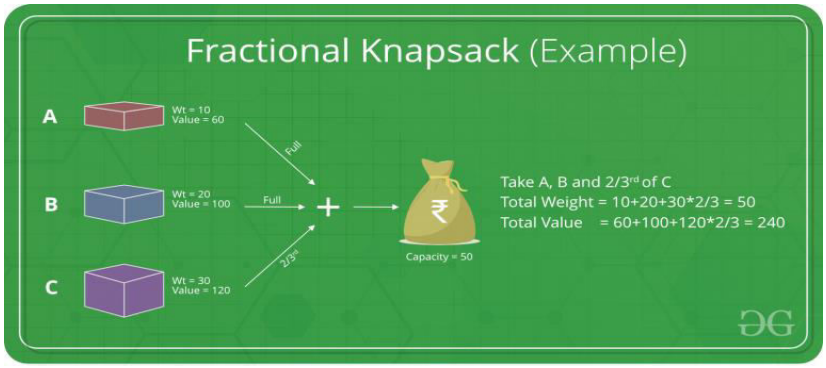
- An algorithm is designed to achieve optimum solution for a given problem.
- In greedy algorithm approach, decisions are made from the given solution domain.
- As being greedy, the closest solution that seems to provide an optimum solution is chosen.
- Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions.

Examples

Most networking algorithms use the greedy approach. Here is a list of few of them −

- Travelling Salesman Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm
- Graph - Map Coloring
- Graph - Vertex Cover
- Knapsack Problem
- Job Scheduling Problem

There are lots of similar problems that uses the greedy approach to find an optimum solution.

- 

Knapsack Problem:

Knapsack is basically means bag. A bag of given capacity.

We want to pack n items in your luggage.

- The ith item is worth v_i dollars and weight w_i pounds.
- Take as valuable a load as possible, but cannot exceed W pounds.
- v_i w_i W are integers.
- W ≤ capacity
- Value ← Max

Input:

- Knapsack of capacity
- List (Array) of weight and their corresponding value.

Output:

- To maximize profit and minimize weight in capacity.
- The knapsack problem where we have to pack the knapsack with maximum value in such a manner that the total weight of the items should not be greater than the capacity of the knapsack.

Knapsack problem can be further divided into two parts:

1. **Fractional Knapsack:** Fractional knapsack problem can be solved by **Greedy Strategy** where as 0 /1 problem is not.
2. It cannot be solved by **Dynamic Programming Approach.**

0/1 Knapsack Problem:

- In this item cannot be broken which means thief should take the item as a whole or should leave it. That's why it is called **0/1 knapsack Problem**.
- Each item is taken or not taken.
- Cannot take a fractional amount of an item taken or take an item more than once.

- It cannot be solved by the Greedy Approach because it is enable to fill the knapsack to capacity.
- **Greedy Approach** doesn't ensure an Optimal Solution.

Example of 0/1 Knapsack Problem:

Example: The maximum weight the knapsack can hold is W is 11. There are five items to choose from. Their weights and values are presented in the following table:

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$												
$w_2 = 2\ v_2 = 6$												
$w_3 = 5\ v_3 = 18$												
$w_4 = 6\ v_4 = 22$												
$w_5 = 7\ v_5 = 28$												

- The [i, j] entry here will be V [i, j], the best value obtainable using the first "i" rows of items if the maximum capacity were j. We begin by initialization and first row.

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0											
$w_3 = 5\ v_3 = 18$	0											
$w_4 = 6\ v_4 = 22$	0											
$w_5 = 7\ v_5 = 28$	0											

- **$V [i, j] = \max \{V [i - 1, j], v_i + V [i - 1, j - w_i]$**

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0											
$w_4 = 6\ v_4 = 22$	0											
$w_5 = 7\ v_5 = 28$	0											

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0											
$w_5 = 7\ v_5 = 28$	0											

The value of V [3, 7] was computed as follows:

$V [3, 7] = \max \{V [3 - 1, 7], v_3 + V [3 - 1, 7 - w_3]$

$= \max \{V [2, 7], 18 + V [2, 7 - 5]\}$

$= \max \{7, 18 + 6\}$

$= 24$

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7\ v_5 = 28$	0											

Finally, the output is:

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7\ v_5 = 28$	0	1	6	7	7	18	22	28	29	34	35	40

- **The maximum value of items in the knapsack is 40, the bottom-right entry).**

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

- **https://www.geeksforgeeks.org/greedy-algorithms/**
- **https://en.wikipedia.org/wiki/Greedy_algorithm**
- **https://www.youtube.com/watch?v=oTTzNMHM05I**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -327

Course Faculty

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 - 42

LECTURE HANDOUTS

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 26.11.2020

Topic of Lecture: Dynamic Programming

Introduction : (Maximum 5 sentences)

- Dynamic Programming (DP) is **an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems** and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Algorithm
- Types of Algorithm

Detailed content of the Lecture:

- Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used.
- Mostly, these algorithms are used for optimization.
- Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems.
- The solutions of sub-problems are combined in order to achieve the best solution.

So we can say that −

- The problem should be able to be divided into smaller overlapping sub-problem.
- An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
- Dynamic algorithms use Memoization.

Comparison

- In contrast to divide and conquer algorithms, where solutions are combined to achieve an overall solution, dynamic algorithms use the output of a smaller sub-problem and then try to

optimize a bigger sub-problem.

- Dynamic algorithms use Memorization to remember the output of already solved sub-problems.

Example

The following computer problems can be solved using dynamic programming approach −

- Fibonacci number series
- Knapsack problem
- Tower of Hanoi
- All pair shortest path by Floyd-Warshall
- Shortest path by Dijkstra
- Project scheduling

Dynamic programming can be used in **both top-down and bottom-up manner.**

Dynamic programming approach

The following are the steps that the dynamic programming follows:

- It breaks down the complex problem into simpler subproblems.
- It finds the optimal solution to these sub-problems.
- It stores the results of subproblems (memoization). The process of storing the results of subproblems is known as memorization.
- It reuses them so that same sub-problem is calculated more than once.
- Finally, calculate the result of the complex problem.

The above five steps are the basic steps for dynamic programming. The dynamic programming is applicable that are having properties such as:

- Those problems that are having overlapping subproblems and optimal substructures.
- Here, optimal substructure means that the solution of optimization problems can be obtained by simply combining the optimal solution of all the subproblems.
- In the case of dynamic programming, the space complexity would be increased as we are storing the intermediate results, but the time complexity would be decreased.

Advantages

- It is very easy to understand and implement.
- It solves the subproblems only when it is required.
- It is easy to debug.

Disadvantages

- It uses the recursion technique that occupies more memory in the call stack. Sometimes when the recursion is too deep, the stack overflow condition will occur.
- It occupies more memory that degrades the overall performance.

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

- **https://www.geeksforgeeks.org/dynamic-programming/**
- **https://www.tutorialspoint.com/data_structures_algorithms/dynamic_programming.htm**
- **https://www.javatpoint.com/dynamic-programming-introduction**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -299

Course Faculty

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

L - 43

LECTURE HANDOUTS

MCA | **I/I**

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 02.12.2020

Topic of Lecture: Warshall's Algorithm for Finding Transitive Closure

Introduction : (Maximum 5 sentences)

- Warshall's algorithm is used to determine the transitive closure of a directed graph or all paths in a directed graph by using the **adjacency matrix**.
- The $R^{(n)}$ matrix will contain ones if it contains a path between vertices with intermediate vertices from any of the n vertices of a graph.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Algorithm
- Types of Algorithm

Detailed content of the Lecture:

- A sequence of vertices is used to define a path in a simple graph. In the k^{th} matrix ($R^{(k)}$), ($r_{ij}^{(k)}$), the element's definition at the i^{th} row and j^{th} column will be one if it contains a path from v_i to v_j.
- For all intermediate vertices, w_q is among the first k vertices that mean $1 \leq q \leq k$.
- The $R^{(0)}$ matrix is used to describe the path without any intermediate vertices.
- So we can say that it is an adjacency matrix.
- The $R^{(n)}$ matrix will contain ones if it contains a path between vertices with intermediate vertices from any of the n vertices of a graph. So we can say that it is a transitive closure.
- Now we will assume a case in which rij(k) is 1 and rij(k-1) is 0. This condition will be true only if it contains a path from vi to vj using the vk. More specifically, the list of vertices is in the following form

v_i, w_q (where $1 \leq q < k$), v_k. w_q (where $1 \leq q < k$), v_j

The above case will be occur only if $r_{ik}^{(k-1)} = r_{kj}^{(k-1)} = 1$. Here, k is subscript.

The $r_{ij}^{(k)}$ will be one if and only if $r_{ij}^{(k-1)} = 1$.

So in summary, we can say that

```
r_ij^(k) = r_ij^(k-1) or (r_ik^(k-1) and r_kj^(k-1))
```

Now we will describe the algorithm of Warshall's Algorithm for computing transitive closure

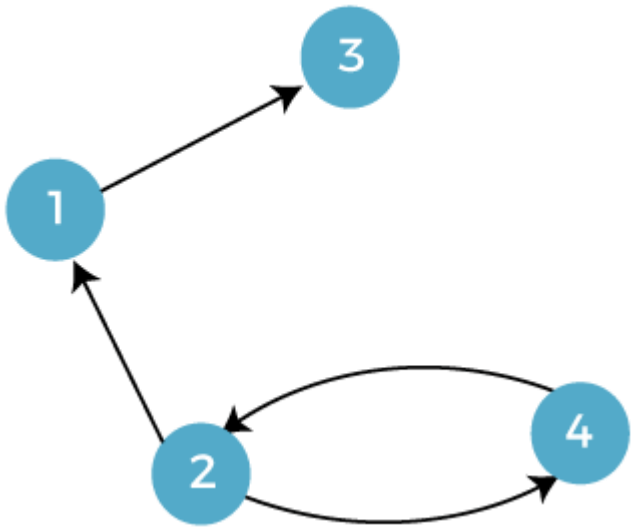
```
Warshall(A[1...n, 1...n]) // A is the adjacency matrix
R^(0) ← A
for k ← 1 to n do
for i ← 1 to n do
for j ← to n do
R^(k)[i, j] ← R^(k-1)[i, j] or (R^(k-1)[i, k] and R^(k-1)[k, j])
return R^(n)
```

Here,

- Time efficiency of this algorithm is (n^3)
- In the Space efficiency of this algorithm, the matrices can be written over their predecessors.
- $\Theta(n^3)$ is the worst-case cost. We should know that the brute force algorithm is better than Warshall's algorithm. In fact, the brute force algorithm is also faster for a space graph.

Example of Transitive closure

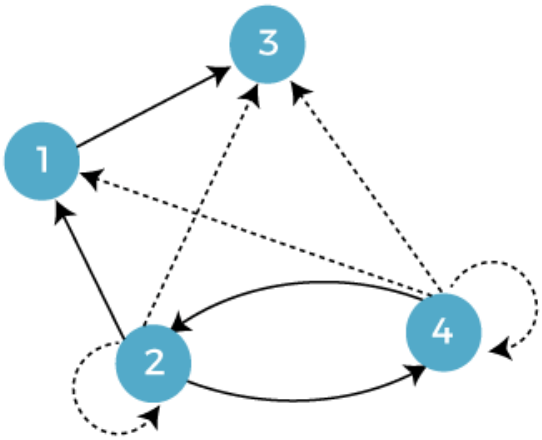
In this example, we will consider two graphs. The first graph is described as follows:



The matrix of this graph is described as follows:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The second graph is described as follows:



The matrix of this graph is described as follows:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

The main idea of these graphs is described as follows:

- The vertices i, j will be contained a path if
- The graph contains an edge from i to j; or
- The graph contains a path from i to j with the help of vertex 1; or
- The graph contains a path from i to j with the help of vertex 1 and/or vertex 2; or
- The graph contains a path from i to j with the help of vertex 1, 2, and/or vertex 3; or
- The graph contains a path from i to j with the help of any other vertices.

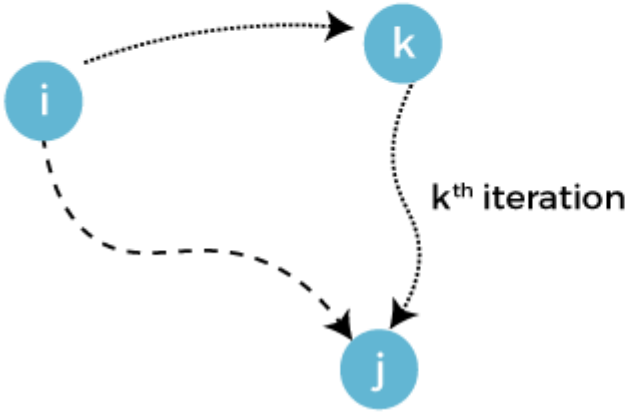
On the kth iteration, the algorithm will use the vertices among 1, …., k, known as the

intermediate, and find out that there is a path exists between i and j vertices or not

$$R^{(k)}[i,j] = \begin{cases} R^{(k-1)}[i,j] \\ \quad \text{or} \\ R^{(k-1)}[i,j] \text{ and } R^{(k-1)}[k,j]) \end{cases}$$

(path using just 1,.....,k-1)

(path from i to k and from k to i using just 1 ,.....,k-1)



- In a directed graph, the **transitive closure** with n vertices is used to describe the n-by-n Boolean matrix T.
- Where, elements in the ith row and jth column will be 1. This can occur only if it contains a directed path form ith vertex to jth vertex.
- Otherwise, the element will be zero. The transitive closure is described as follows:

$$T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

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

- **https://www.javatpoint.com/warshalls-algorithm**
- **https://www.geeksforgeeks.org/transitive-closure-of-a-graph/**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -304

Course Faculty

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

LECTURE HANDOUTS

L - 44

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 07.12.2020

Topic of Lecture: Backtracking - Sum of Subset Problem

Introduction : (Maximum 5 sentences)

- Subset sum problem is **the problem of finding a subset such that the sum of elements equal a given number**. The backtracking approach generates all permutations in the worst case but in general, performs better than the recursive approach towards subset sum problem.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Algorithm
- Types of Algorithm

Detailed content of the Lecture:

- In this problem, there is a given set with some integer elements. And another some value is also provided, we have to find a subset of the given set whose sum is the same as the given sum value.
- Here backtracking approach is used for trying to select a valid subset when an item is not valid, we will backtrack to get the previous subset and add another element to get the solution.

Input and Output

```
Input:
This algorithm takes a set of numbers, and a sum value.
The Set: {10, 7, 5, 18, 12, 20, 15}
The sum Value: 35
Output:
All possible subsets of the given set, where sum of each element for
every subsets is same as the given sum value.
{10,  7,  18}
{10,  5,  20}
{5,  18,  12}
```

```
{20,  15}
```

Algorithm

subsetSum(set, subset, n, subSize, total, node, sum)

Input − The given set and subset, size of set and subset, a total of the subset, number of elements in the subset and the given sum.

Output − All possible subsets whose sum is the same as the given sum.

```
Begin
  if total = sum, then
    display the subset
    //go for finding next subset
    subsetSum(set, subset, , subSize-1, total-set[node], node+1, sum)
    return
  else
    for all element i in the set, do
      subset[subSize] := set[i]
      subSetSum(set, subset, n, subSize+1, total+set[i], i+1, sum)
    done
End
```

Example

```
#include <iostream>
using namespace std;

void displaySubset(int subSet[], int size) {
  for(int i = 0; i < size; i++) {
    cout << subSet[i] << "  ";
  }
  cout << endl;
}

void subsetSum(int set[], int subSet[], int n, int subSize, int total, int nodeCount ,int sum) {
  if( total == sum) {
    displaySubset(subSet, subSize);     //print the subset
    subsetSum(set,subSet,n,subSize-1,total-set[nodeCount],nodeCount+1,sum);     //for other subsets
    return;
  }else {
```

```
    for( int i = nodeCount; i < n; i++ ) {      //find node along breadth
      subSet[subSize] = set[i];
      subsetSum(set,subSet,n,subSize+1,total+set[i],i+1,sum);     //do for next node in depth
    }
  }
}

void findSubset(int set[], int size, int sum) {
  int *subSet = new int[size];     //create subset array to pass parameter of subsetSum
  subsetSum(set, subSet, size, 0, 0, 0, sum);
  delete[] subSet;
}

int main() {
  int weights[] = {10, 7, 5, 18, 12, 20, 15};
  int size = 7;
  findSubset(weights, size, 35);
}
```

Output

```
10   7  18
10   5  20
5   18  12
20  15
```

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

- **https://www.geeksforgeeks.org/subset-sum-backtracking-4/**
- **https://www.tutorialspoint.com/Subset-Sum-Problem**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -396

Course Faculty

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 - 45

LECTURE HANDOUTS

MCA

I/I

Course Name with Code : **19CAB02 - Data Structures and Algorithms**

Course Faculty : **Mrs.G.Krishnaveni**

Unit : **Algorithm Design and Analysis**

Date of Lecture: 11.12.2020

Topic of Lecture: Branch and Bound - Travelling Salesman Problem.

Introduction : (Maximum 5 sentences)

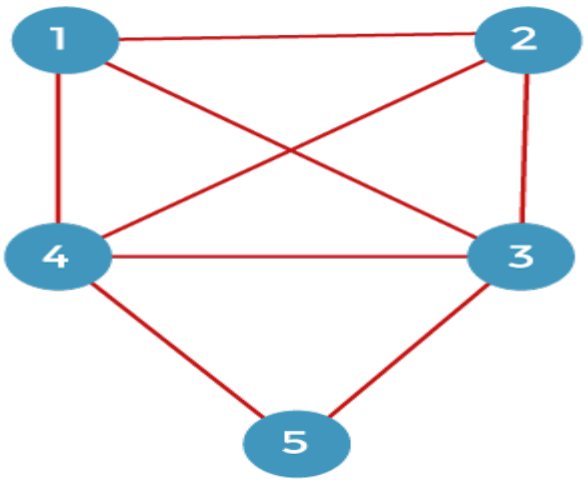
- To find the shortest possible tour that visits every city exactly once and returns to the starting point.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Algorithm
- Types of Algorithm

Detailed content of the Lecture:

- Given the vertices, the problem here is that we have to travel each vertex exactly once and reach back to the starting point. Consider the below graph:



- As we can observe in the above graph that there are 5 vertices given in the graph.
- We have to find the shortest path that goes through all the vertices once and returns back to the starting vertex. We mainly consider the starting vertex as 1, then traverse through the vertices 2, 3, 4, and 5, and finally return to vertex 1.

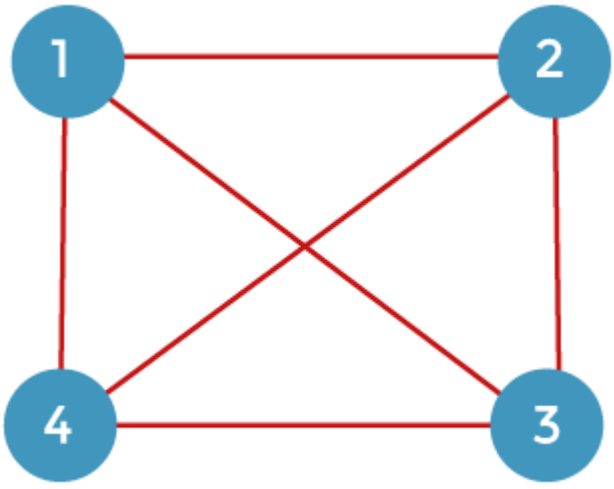
The adjacent matrix of the problem is given below:

	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	30	10	11
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

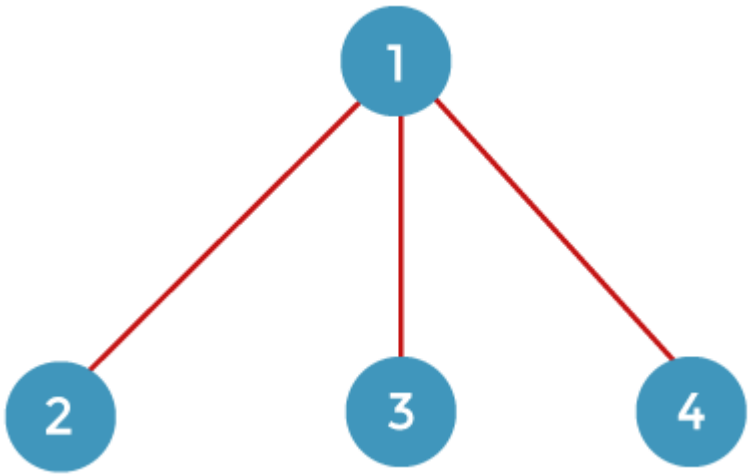
-

Let's first understand the approach then we solve the above problem.

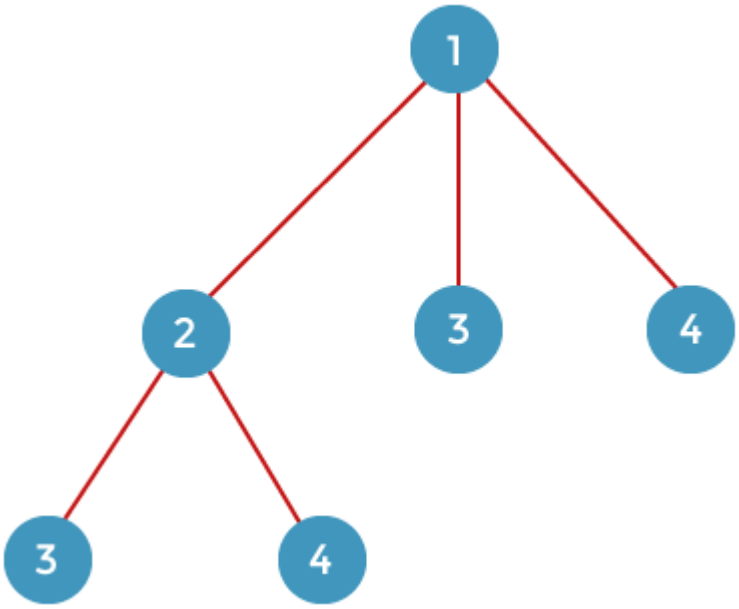
The graph is given below, which has four vertices:



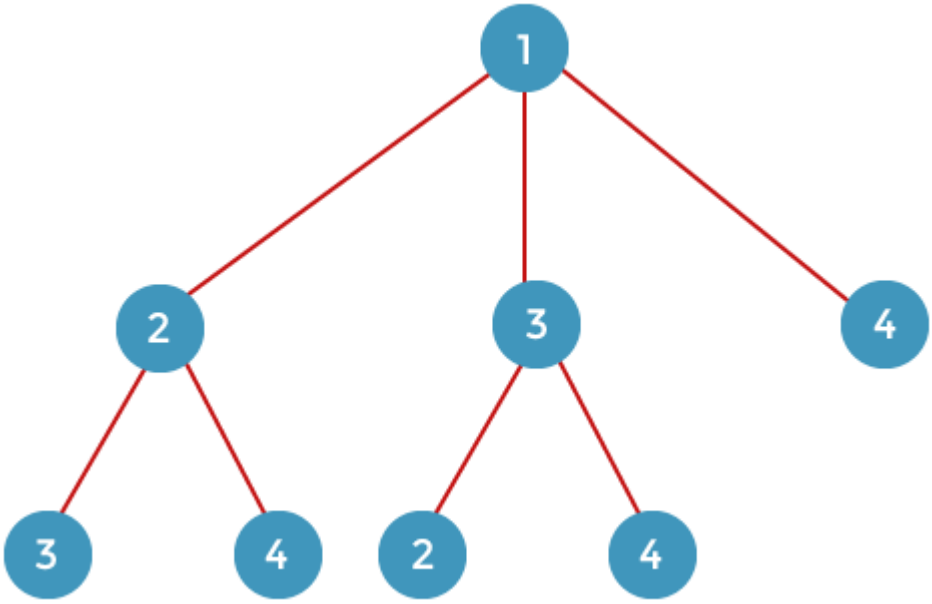
Suppose we start travelling from vertex 1 and return back to vertex 1. There are various ways to travel through all the vertices and returns to vertex 1. We require some tools that can be used to minimize the overall cost. To solve this problem, we make a state space tree. From the starting vertex 1, we can go to either vertices 2, 3, or 4, as shown in the below diagram.



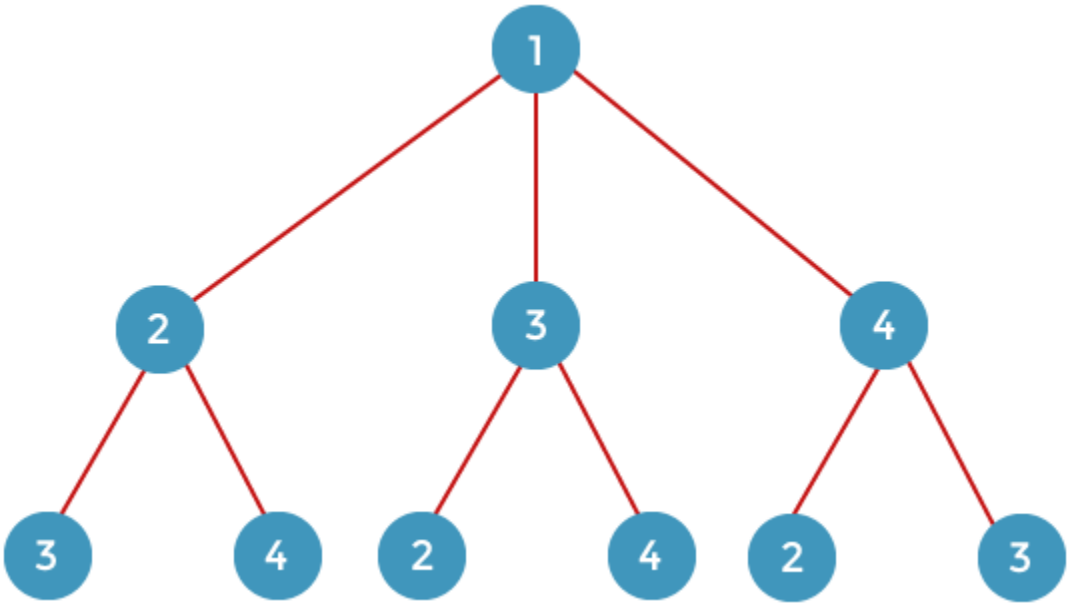
From vertex 2, we can go either to vertex 3 or 4. If we consider vertex 3, we move to the remaining vertex, i.e., 4. If we consider the vertex 4 shown in the below diagram:



From vertex 3, we can go to the remaining vertices, i.e., 2 or 4. If we consider the vertex 2, then we move to remaining vertex 4, and if we consider the vertex 4 then we move to the remaining vertex, i.e., 3 shown in the below diagram:



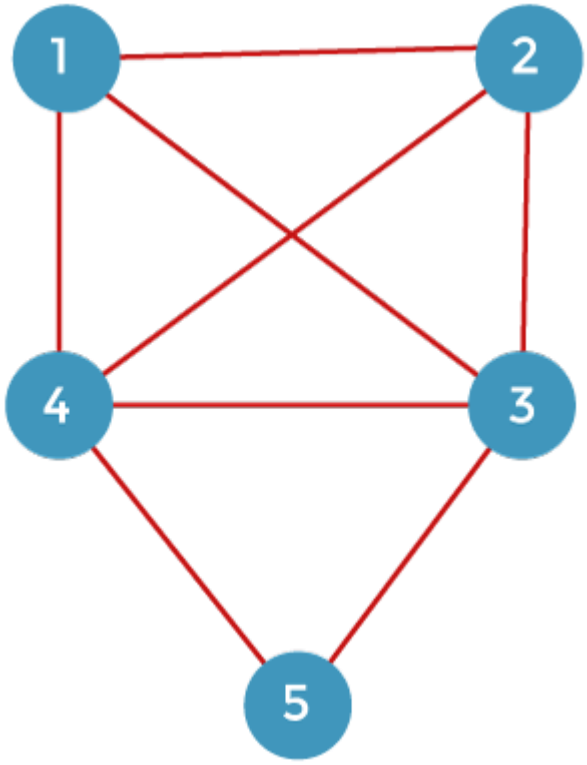
From vertex 4, we can go to the remaining vertices, i.e., 2 or 3. If we consider vertex 2, then we move to the remaining vertex, i.e., 3, and if we consider the vertex 3, then we move to the remaining vertex, i.e., 2 shown in the below diagram:



The above is the complete state space tree. The state space tree shows all the possibilities. Backtracking and branch n bound both use the state space tree, but their approach to solve the problem is different. Branch n bound is a better approach than backtracking as it is more efficient. In order to solve the problem using branch n bound, we use a level order. First, we will observe in which order, the nodes are generated. While creating the node, we will calculate the cost of the node simultaneously. If we find the cost of any node greater than the upper bound, we will remove that node. So, in this case, we will generate only useful nodes

but not all the nodes.

Let's consider the above problem.



	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	30	10	11
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

As we can observe in the above adjacent matrix that 10 is the minimum value in the first row, 2 is the minimum value in the second row, 2 is the minimum value in the third row, 3 is the minimum value in the third row, 3 is the minimum value in the fourth row, and 4 is the minimum value in the fifth row.

Now, we will reduce the matrix. We will subtract the minimum value with all the elements of a row. First, we evaluate the first row. Let's assume two variables, i.e., i and j, where 'i' represents the rows, and 'j' represents the columns.

When i = 0, j =0

M[0][0] = ∞-10= ∞

When i = 0, j = 1

M[0][1] = 20 - 10 = 10

When i = 0, j = 2

M[0][2] = 30 - 10 = 20

When i = 0, j = 3

M[0][3] = 10 - 10 = 0

When i = 0, j = 4

M[0][4] = 11 - 10 = 1

The matrix is shown below after the evaluation of the first row:

	0	1	2	3	4
0	∞	10	20	0	1
1	13	∞	14	2	0
2	3	5	∞	2	4
3	19	6	18	∞	3
4	16	4	7	16	∞

Consider the second row.

When i = 1, j =0

M[1][0] = 15-2= 13

When i = 1, j = 1

M[1][1] = ∞ - 2= ∞

When i = 1, j = 2

M[1][2] = 16 - 2 = 14

When i = 1, j = 3

M[1][3] = 4 - 2 = 2

When i = 1, j = 4

M[1][4] = 2 - 2 = 0

The matrix is shown below after the evaluation of the second row:

	0	1	2	3	4
0	∞	10	20	0	1
1	13	∞	14	2	0
2	1	3	∞	0	2
3	19	6	18	∞	3
4	16	4	7	16	∞

Consider the third row:

When i = 2, j =0

M[2][0] = 3-2= 1

When i = 2, j = 1

M[2][1] = 5 - 2= 3

When i = 2, j = 2

M[2][2] = ∞ - 2 = ∞

When i = 2, j = 3

M[2][3] = 2 - 2 = 0

When i = 2, j = 4

M[2][4] = 4 - 2 = 2

The matrix is shown below after the evaluation of the third row:

Consider the fourth row:

When i = 3, j =0

M[3][0] = 19-3= 16

When i = 3, j = 1

M[3][1] = 6 - 3= 3

When i = 3, j = 2

M[3][2] = 18 - 3 = 15

When i = 3, j = 3

M[3][3] = ∞ - 3 = ∞

When i = 3, j = 4

M[3][4] = 3 - 3 = 0

The matrix is shown below after the evaluation of the fourth row:

	0	1	2	3	4
0	∞	10	20	0	1
1	13	∞	14	2	0
2	1	3	∞	0	2
3	16	3	15	∞	0
4	16	4	7	16	∞

Consider the fifth row:

When i = 4, j =0

M[4][0] = 16-4= 12

When i = 4, j = 1

M[4][1] = 4 - 4= 0

When i = 4, j = 2

M[4][2] = 7 - 4 = 3

When i = 4, j = 3

M[4][3] = 16 - 4 = 12

When i = 4, j = 4

M[4][4] = ∞ - 4 = ∞

The matrix is shown below after the evaluation of the fifth row:

	0	1	2	3	4
0	∞	10	20	0	1
1	13	∞	14	2	0
2	1	3	∞	0	2
3	16	3	15	∞	0
4	12	0	3	12	∞

The above matrix is the reduced matrix with respect to the rows.

Now we reduce the matrix with respect to the columns. Before reducing the matrix, we first find the minimum value of all the columns. The minimum value of first column is 1, the minimum value of the second column is 0, the minimum value of the third column is 3, the minimum value of the fourth column is 0, and the minimum value of the fifth column is 0, as shown in the below matrix:

Now we reduce the matrix.

Consider the first column.

When i = 0, j =0

M[0][0] = ∞-1= ∞

When i = 1, j = 0

M[1][0] = 13 - 1= 12

When i = 2, j = 0

M[2][0] = 1 - 1 = 0

When i = 3, j = 0

M[3][0] = 16 - 1 = 15

When i = 4, j = 0

M[4][0] = 12 - 1 = 11

The matrix is shown below after the evaluation of the first column:

	0	1	2	3	4
0	∞	10	20	0	1
1	12	∞	14	2	0
2	0	3	∞	0	2
3	15	3	15	∞	0
4	11	0	3	12	∞

Since the minimum value of the first and the third columns is non-zero, we will evaluate only first and third columns. We have evaluated the first column. Now we will evaluate the third column.

Consider the third column.

When i = 0, j =2

M[0][2] = 20-3= 17

When i = 1, j = 2

M[1][2] = 13 - 1= 12

When i = 2, j = 2

M[2][2] = 1 - 1 = 0

When i = 3, j = 2

M[3][2] = 16 - 1 = 15

When i = 4, j = 2

M[4][2] = 12 - 1 = 11

The matrix is shown below after the evaluation of the third column:

	0	1	2	3	4
0	∞	10	17	0	1
1	12	∞	12	2	0
2	0	3	0	0	2
3	15	3	15	∞	0
4	11	0	0	12	∞

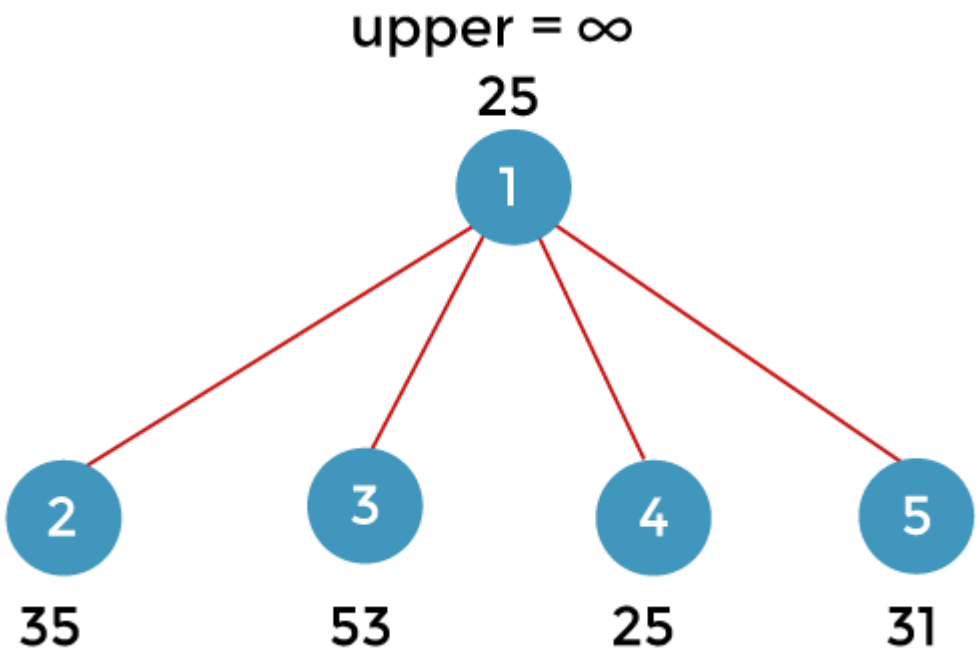
The above is the reduced matrix. The minimum value of rows is 21, and the columns is 4. Therefore, the total minimum value is (21 + 4) equals to 25.

Let's understand that how to solve this problem using branch and bound with

the help of a state-space tree.

- To make a state-space tree, first, we consider node 1.
- From node 1, we can go either to nodes 2, 3, 4, or 5 as shown in the below image.
- The cost of node 1 would be the cost which we achieved in the above-reduced matrix, i.e.., 25.
- Here, we will also maintain the upper bound. Initially, the upper bound would-be

infinity.



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

- https://www.javatpoint.com/traveling-salesperson-problem-using-branch-and-bound
- **https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2**

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

- Anany Levitin,"The Design of Analysis of Algorithms" Pearson education, Page No -406
-

Course Faculty

Verified by HOD