MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-1

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher :M.S.Soundarya

Unit : I -Introduction **Date of Lecture:**

Topic of Lecture: Introduction

Introduction :

- Data structure defines a way of organizing all data items that consider not only the elements stored but also stores the relationship between the elements.
- Data structure can divide with two types: linear and non-linear structure.

Prerequisite knowledge for Complete understanding and learning of Topic:

- One Programming language (C/C++)
- Knowledge on array, pointer and recursion

Detailed content of the Lecture:

Data Structure: An arrangement of data in a computer's memory or Way of organizing data in a computer so that it can be used effectively. In general data structure types include the file, array, record, table, tree.etc..

Classification of Data Structure:

- ✓Static data structures.: A data structure formed when the number of data items are known in advance is referred as static data structure or fixed size data structure..EX: Array, pointers, structures, etc.,
- ✓Dynamic data structures: A data structure formed when the number of data items are not known in advance is known as dynamic data structure or variable size data structure. Ex: linked lists, stacks, queues, trees

Types of data structure

- ✓ Linear data structure.
- ✓ Non-linear data structure

Linear data structures:

Linear data structures, where the data are arranged in sequential order or the data having a linear relationship between its adjacent elements. Linked lists are examples of linear data structures.eg linked list, array

Anu	Abirami	Arun	Babu	Balaji	Gowtham

Non-linear data structures:

Non-linear data structures are data structures that don't have a linear relationship between its adjacent elements but have a hierarchical relationship between the elements. Trees and graphs are examples of non-linear data structures. Eg: Tree , Graph

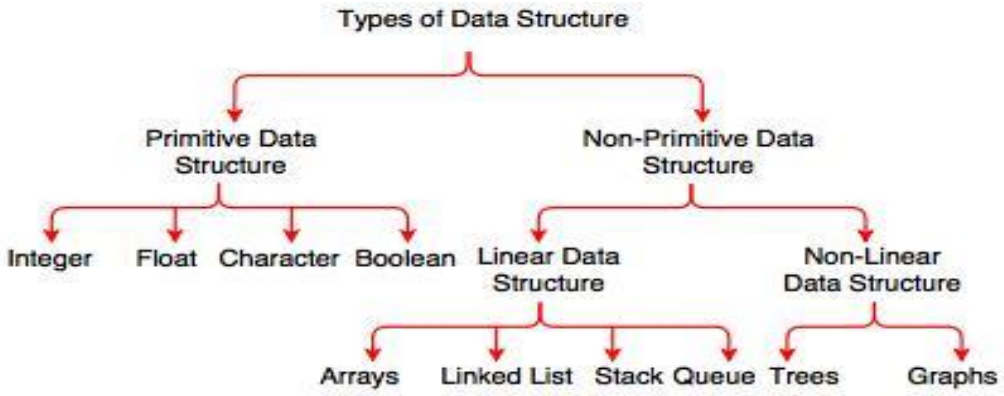


Fig. Types of Data Structure

Primitive Data Types

Each variable has a specific data typeIt tells - size, range and the type of a value that can be stored in a variable.

There are 4 basic primitive data types

- integer data types, such as short, int, long
- floating-point data types, such as float, double
- character data type, such as char
- Pointer

Abstract Data Type

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

Basic Terminologies :

- ✓ **Data:** Data are simply values or sets of values.
- ✓ **Data items:** Data items refers to a single unit of values.
- ✓ Data items that are divided into sub-items are called Group items. Ex: An Employee Name may be divided into three subitems- first name, middle name, and last name. Data items that are not able to divide into sub-items are called Elementary items. Ex: SSN
- ✓ **Entity:** An entity is something that has certain attributes or properties which may be assigned values. The values may be either numeric or non-numeric. Ex: Attributes- Names, Age, Sex, SSN Values- Rohland Gail, 34, F, 134-34-5533 Entities with similar attributes form an entity set.
    - Each attribute of an entity set has a range of values, the set of all possible values that could be assigned to the particular attribute. The term "information" is sometimes used for data with given attributes, of, in other words meaningful or processed data.

- ✓ **Field:** Field is a single elementary unit of information representing an attribute of an entity.
- ✓ **Record:** Record is the collection of field values of a given entity.
- ✓ **File:** File is the collection of records of the entities in a given entity set.

Applications of data structures

- ✓ Arranging similar types of books in a library based on their titles.
- ✓ Noise removal filters in an audio recording use arrays.
- ✓ Two-dimensional arrays which are generally called Matrices are mainly used in image processing.
- ✓ In Speech Processing, each speech signal is an array of Signal amplitude.
- ✓ A question Paper is also an array of numbered questions that are assigned with some marks.
- ✓ Playfair-cipher, an old algorithm used for encryption also uses a 2D array of alphabets as key to encrypt/decrypt text

Video Content / Details of website for further learning (if any):
https://youtu.be/zWg7U0OEAoE
https://nptel.ac.in/courses/106/102/106102064/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C, Pearson India,2012- **page nos:** 1-6

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L- 2

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher :M.S.Soundarya

Unit : I -Introduction **Date of Lecture:**

Topic of Lecture: Data Structure Operations

Introduction :

Data structure is a representation of data and the operations allowed on that data.It is a way to store and organize data in order to facilitate the access and modifications.

Data Structure are the method of representing of logical relationships between individual data elements related to the solution of a given problem.

- Insertion
- Deletion
- Traversal

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basic Terminologies
- Elementary Data Organizations
- Data Structures

Detailed content of the Lecture:

- Insertion:
- Insertion means addition of a new data element in a data structure.

Deletion:

- Deletion means removal of a data element from a data structure if it is found.

Searching:

- Searching involves searching for the specified data element in a data structure.

Traversal:

- Traversal of a data structure means processing all the data elements present in it.

Sorting:

- Arranging data elements of a data structure in a specified order is called sorting.

Merging:

- Combining elements of two similar data structures to form a new data structure of the same type, is called merging

Advantages

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

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

https://nptel.ac.in/courses/106104128/
https://www.youtube.com/watch?v=6vT1EoPYpTQ

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

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

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-3

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher :M.S.Soundarya

Unit : I -Introduction **Date of Lecture:**

Topic of Lecture: Abstract Data Types ADT

Introduction :

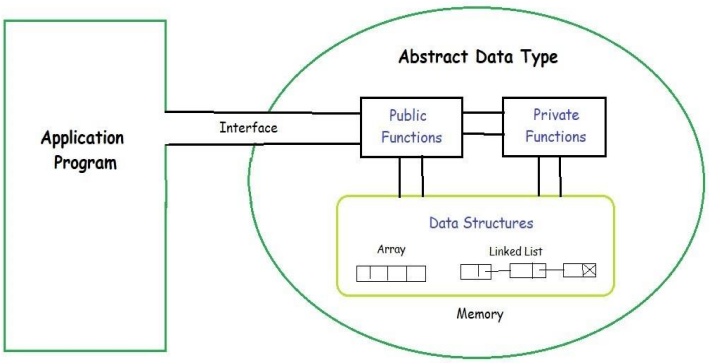
Abstract Data type (ADT) is a type (or class) for objects whose behaviour is defined by a set of value and a set of operations.

Prerequisite knowledge for Complete understanding and learning of Topic:

1.Basics of Algorithm
2.Programming Knowledge
3.Data Structure
4.Mathematical knowledge

Detailed content of the Lecture:

- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called “abstract” because it gives an implementation-independent view. The process of providing only the essentials and hiding the details is known as abstraction.



- The user of data type does not need to know how that data type is implemented, for example, we have been using Primitive values like int, float, char data types only with the knowledge that these data type can operate and be performed on without any idea of how they are implemented. So a user only needs to know what a data type can do, but not how it will be implemented. Think of ADT as a black box which hides the inner structure and design of the data type. Now

we'll define three ADTs namely
List ADT,
Stack ADT,
Queue ADT.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/abstract-data-types/#:~:text=Abstract%20Data%20type%20(ADT)%20is,and%20a%20set%20of%20operations.&text=So%20a%20user%20only%20needs,design%20of%20the%20data%20type.

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :77

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L4

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures andFiles

Course Teacher :M.S.Soundarya

Unit : I -Introduction **Date of Lecture:**

Topic of Lecture: List ADT

Introduction :

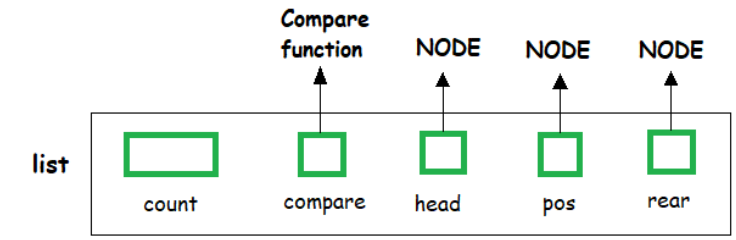
The data is generally stored in key sequence in a list which has a head structure consisting of *count*, *pointers* and *address of compare function* needed to compare the data in the list.

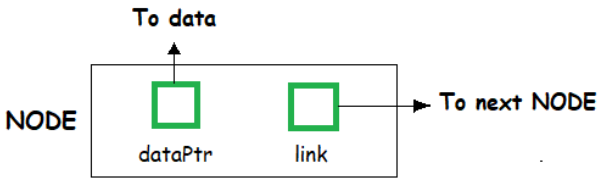
Prerequisite knowledge for Complete understanding and learning of Topic:

1.Basics of Algorithm
2.Programming Knowledge
3.Data Structure
4.Mathematical knowledge

Detailed content of the Lecture:

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

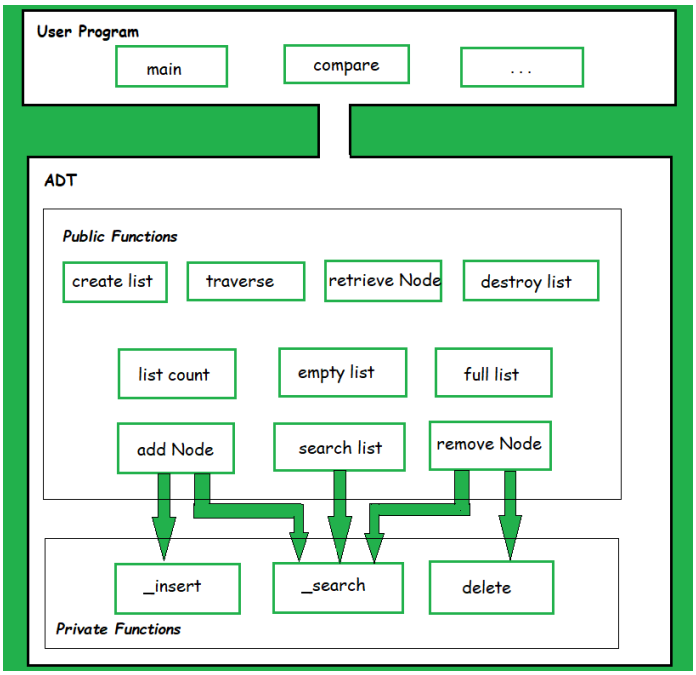




Example:

```
  //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;
```

The **List ADT Functions** is given below



A list contains elements of the same type arranged in sequential order and following operations can be performed on the list.

- get() – Return an element from the list at any given position.
- insert() – Insert an element at any position of the list.
- remove() – Remove the first occurrence of any element from a non-empty list.
- removeAt() – Remove the element at a specified location from a non-empty list.
- replace() – Replace an element at any position by another element.
- size() – Return the number of elements in the list.
- isEmpty() – Return true if the list is empty, otherwise return false.
- isFull() – Return true if the list is full, otherwise return false.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/abstract-data-types/#:~:text=Abstract%20Data%20type%20(ADT)%20is,these%20operations%20will%20be%20implemented.&text=Now%20we'll%20define%20three,%2C%20Stack%20ADT%2C%20Queue%20ADT.
https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/ListADT.html

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :78

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-5

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher :M.S.Soundarya

Unit : I -Introduction **Date of Lecture:**

Topic of Lecture: Array Based Implementation

Introduction :

A list is a sequential data structure, ie. a collection of items accessible one after another beginning at t and ending at the tail.

Prerequisite knowledge for Complete understanding and learning of Topic:

1.Basics of Algorithm
2.Programming Knowledge
3.Data Structure
4.Mathematical knowledge

Detailed content of the Lecture:

It is a widely used data structure for applications which do not need random access Addition and removals can be made at any position in the list

lists are normally in the form of a_1,a_2,a_3.....an. The size of this list is n.The first element of the list is a1,and the last element is an.The position of element ai in a list is i.

List of size 0 is called as null list.

Basic Operations on a List

Creating a list
Traversing the list
Inserting an item in the list
Deleting an item from the list
Concatenating two lists into one

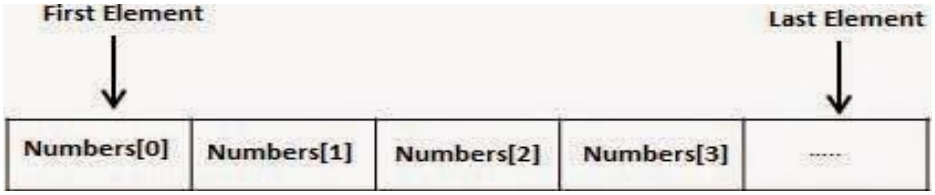
Implementation of List:

A list can be implemented in two ways

1. Array list
2. Linked list

Array

An Array is a data structure which can store a fixed-size sequential collection of elements of the same type.A specific element in an array is accessed by an index.



Different Types of Arrays

- One-dimensional array: only one index is used int [5]
- Multi-dimensional array: array involving more than one index int [2,5]
- Static array: the compiler determines how memory will be allocated for the array
- Dynamic array: memory allocation takes place during execution

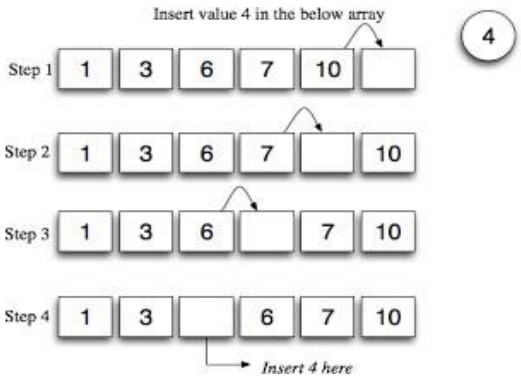
One Dimensional Static Array

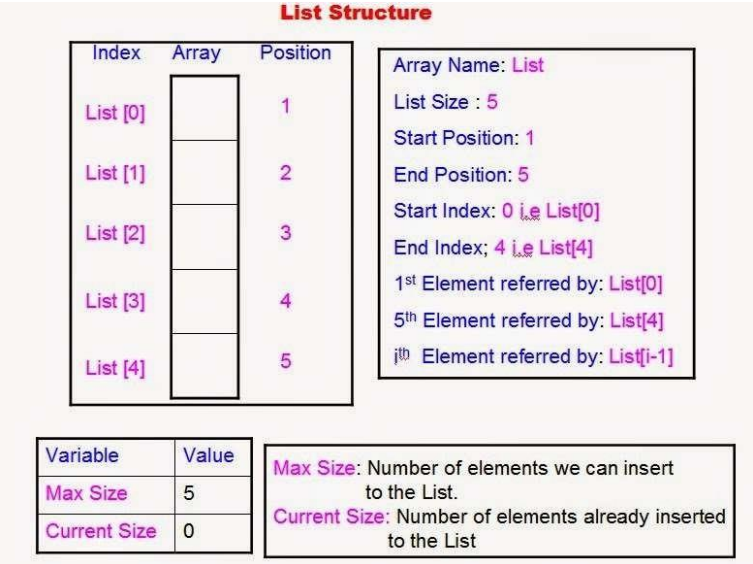
- Syntax:
  - **ElementType arrayName [CAPACITY];**
  - **ElementType arrayName [CAPACITY] = { initializer_list };**

Operations On Lists

- We"ll consider only few operations and not all operations on Lists
- Let us consider Insert
- There are two possibilities:
  - Ordered List
  - Unordered List

Insertion in Detail





OPERATIONS:

1. **Is Empty(LIST)**

If (Current Size== 0) "LIST is Empty“

else "LIST is not Empty“

2. **Is Full(LIST)**

If (Current Size=Max Size) "LIST is FULL"

else "LIST is not FULL“

Video Content / Details of website for further learning (if any):
https://www.brainkart.com/article/Array-Implementation-of-Lists-List_6969/#:~:text=This%20implementation%20stores%20the%20list, the%20size%20of%20the%20list.

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :78

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-6

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher :M.S.Soundarya

Unit : I -Introduction **Date of Lecture:**

Topic of Lecture: Linked List Implementation

Introduction :

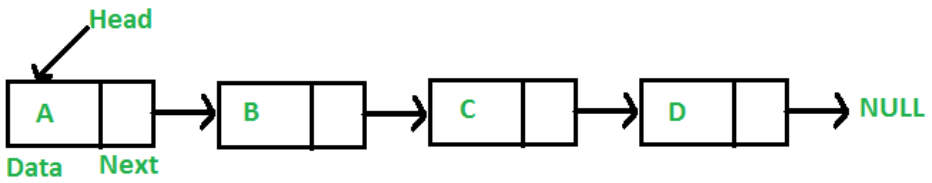
A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.

Prerequisite knowledge for Complete understanding and learning of Topic:

1.Basics of Algorithm
2.Programming Knowledge
3.Data Structure

Detailed content of the Lecture:

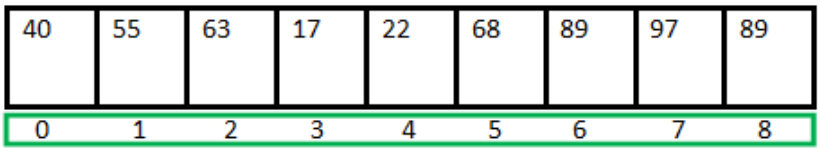
A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



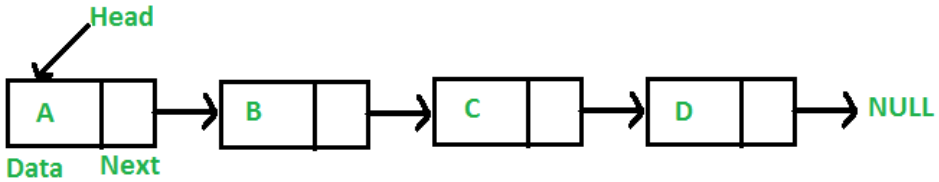

```
class Node {
public:
    int data;
    Node* next;
};
```

Linked List vs Array

Arrays store elements in contiguous memory locations, resulting in easily calculable addresses for the elements stored and this allows faster access to an element at a specific index. Linked lists are less rigid in their storage structure and elements are usually not stored in contiguous locations, hence they need to be stored with additional tags giving a reference to the next element. This difference in the data storage scheme decides which data structure would be more suitable for a given situation.

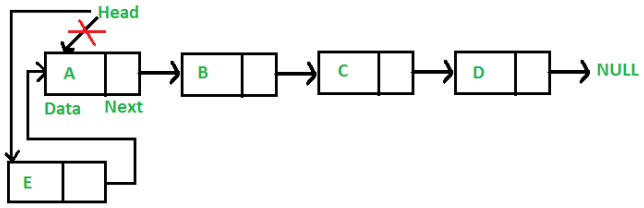


Array Length = 9
First Index = 0
Last Index = 8

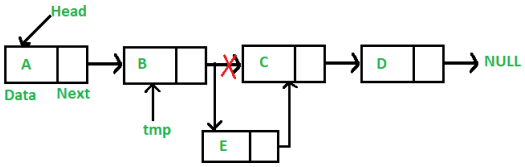


Insertion:

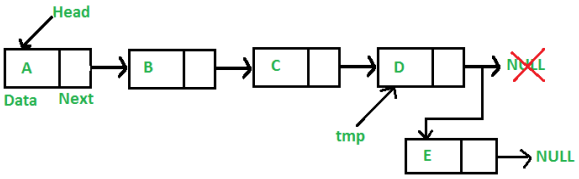
1.Add a node at the front



2.Add a node after a given node



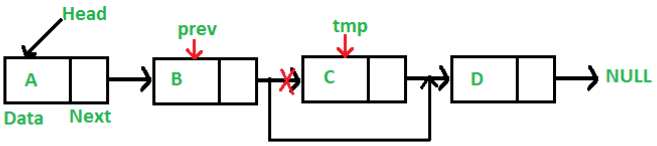
3.Add a node at the end



Deleting a node

Iterative Method:

To delete a node from the linked list, we need to do the following steps.
1) Find the previous node of the node to be deleted.
2) Change the next of the previous node.
3) Free memory for the node to be deleted.



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

https://www.geeksforgeeks.org/data-structures/linked-list/
https://www.geeksforgeeks.org/linked-list-set-1-introduction/

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

Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :79

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

LECTURE HANDOUTS

L - 7

AI&DS

Year/Sem

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : I - Introduction **Date of Lecture:**

Topic of Lecture: Doubly Linked List

Introduction :

A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.

Prerequisite knowledge for Complete understanding and learning of Topic:

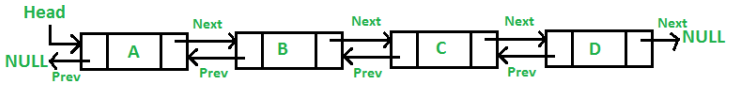
1.Basics of Algorithm

2.Programming Knowledge

3.Data Structure

Detailed content of the Lecture:

A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.




```
/* Node of a doubly linked list */
class Node
{
        public:
        int data;
        Node* next; // Pointer to next node in DLL
        Node* prev; // Pointer to previous node in DLL
};
```

Advantages over singly linked list

1) A DLL can be traversed in both forward and backward direction.

2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

3) We can quickly insert a new node before a given node.

In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

Disadvantages over singly linked list

1) Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though (See this and this).

2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

Insertion

A node can be added in four ways

1) At the front of the DLL

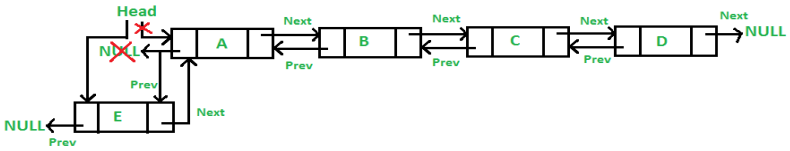
2) After a given node.

3) At the end of the DLL

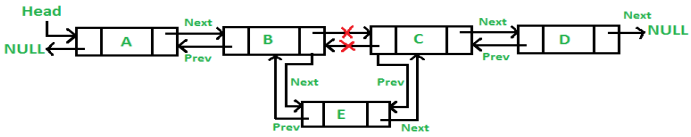
4) Before a given node.

Insertion:

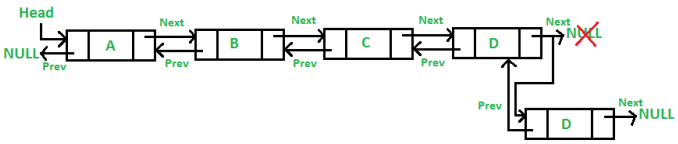
1) Add a node at the front



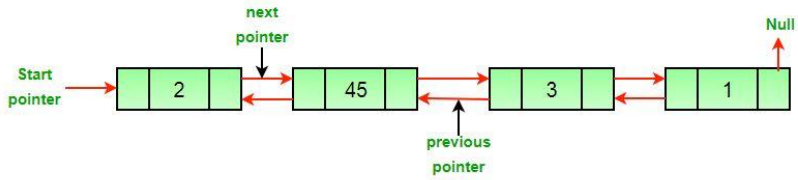
2) Add a node after a given node



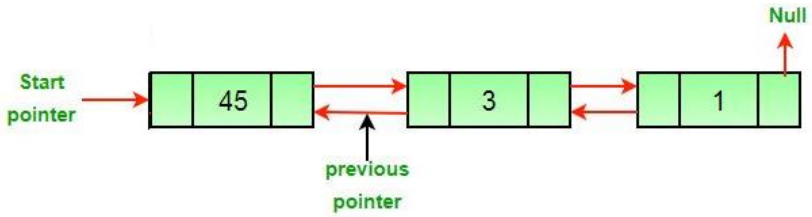
3) Add a node at the end



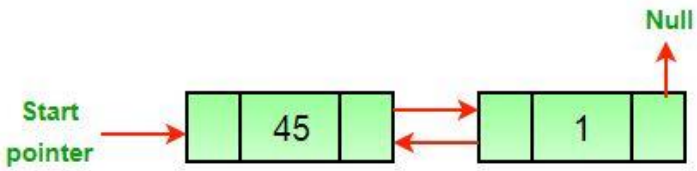
Delete a node in a Doubly Linked List



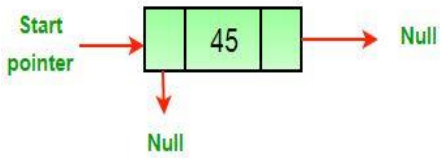
1.After the deletion of the head node.



2. After the deletion of the middle node.



3. After the deletion of the last node



Algorithm

- Let the node to be deleted be *del*.
- If node to be deleted is head node, then change the head pointer to next current head.

if *headnode* == *del* then
    headnode = *del*.nextNode

- Set *next* of previous to *del*, if previous to *del* exists.

if *del*.nextNode != *none*
    del.nextNode.previousNode = *del*.previousNode

- Set *prev* of next to *del*, if next to *del* exists.

if *del*.previousNode != *none*
    del.previousNode.nextNode = *del*.next

Video Content / Details of website for further learning (if any): **https://www.geeksforgeeks.org/delete-a-node-in-a-doubly-linked-list/** **https://www.geeksforgeeks.org/doubly-linked-list/**
Important Books/Journals for further learning including the page nos.: Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :79

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 - 8

AI&DS

Year/Sem

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : I - Introduction **Date of Lecture:**

Topic of Lecture: Circular Linked List

Introduction :

Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.

Prerequisite knowledge for Complete understanding and learning of Topic:

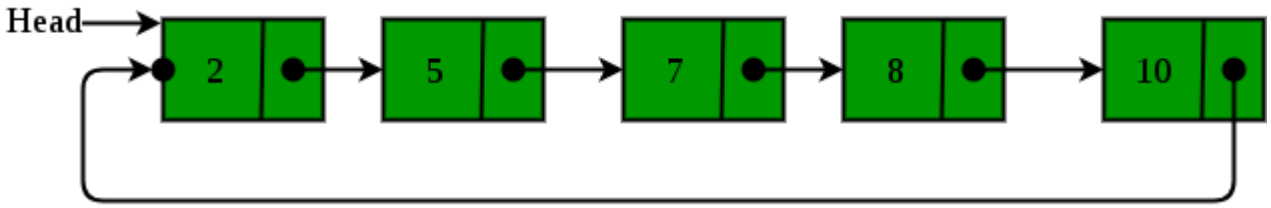
1.Basics of Algorithm

2.Programming Knowledge

3.Data Structure

Detailed content of the Lecture:

Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.



Advantages of Circular Linked Lists:

1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

2) 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.

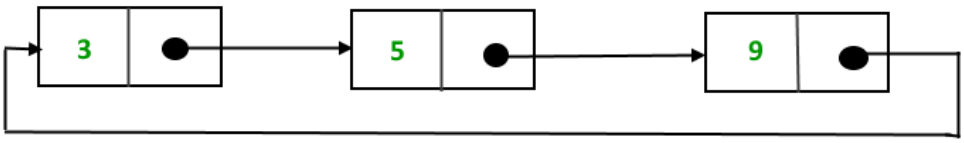
3) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and 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.

4) Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

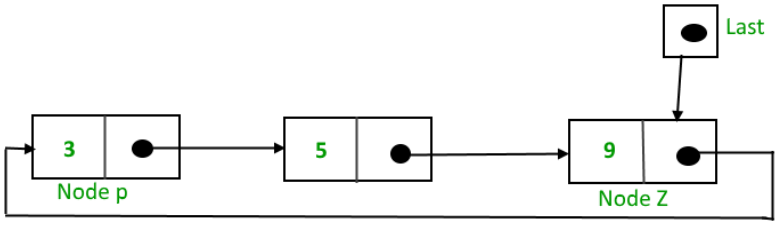
Circular Singly Linked List | Insertion :

The structure thus formed is a circular singly linked list and looks like this:



Implementation

To implement a circular singly linked list, we take an external pointer that points to the last node of the list. If we have a pointer last pointing to the last node, then last -> next will point to the first node.



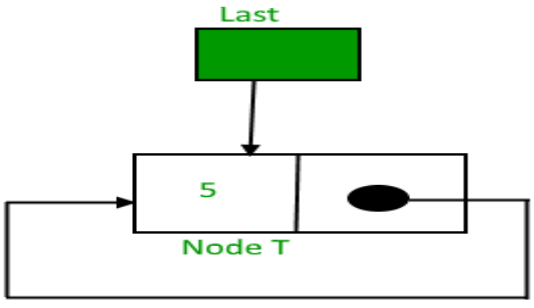
Insertion

A node can be added in three ways:

- Insertion in an empty list
- Insertion at the beginning of the list
- Insertion at the end of the list
- Insertion in between the nodes

Insertion in an empty List

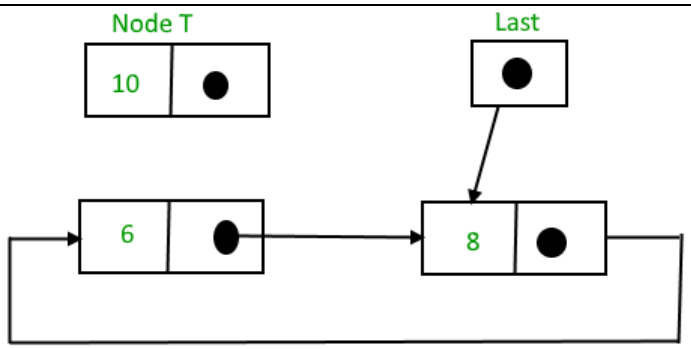
Initially, when the list is empty, the *last* pointer will be NULL.



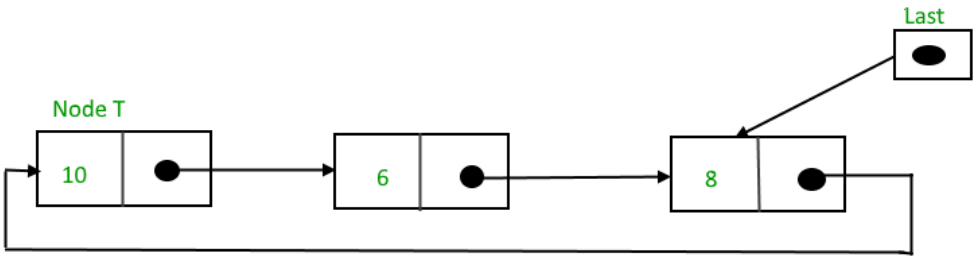
Insertion at the beginning of the list

To insert a node at the beginning of the list, follow these steps:

1. Create a node, say T.
2. Make T -> next = last -> next.
3. last -> next = T.



After insertion,

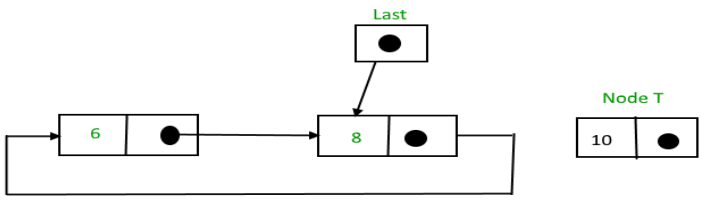



```
struct Node *addBegin(struct Node *last, int data)
{
if (last == NULL)
        return addToEmpty(last, data);

// Creating a node dynamically.
struct Node *temp
                = (struct Node *)malloc(sizeof(struct Node));

// Assigning the data.
temp -> data = data;

// Adjusting the links.
temp -> next = last -> next;
last -> next = temp;

return last;
}
```

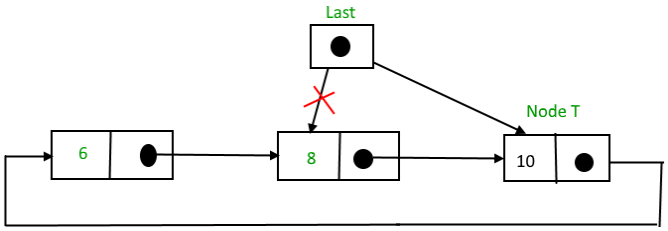
Insertion at the end of the list

To insert a node at the end of the list, follow these steps:

1. Create a node, say T.
2. Make T -> next = last -> next;
3. last -> next = T.
4. last = T.



After insertion



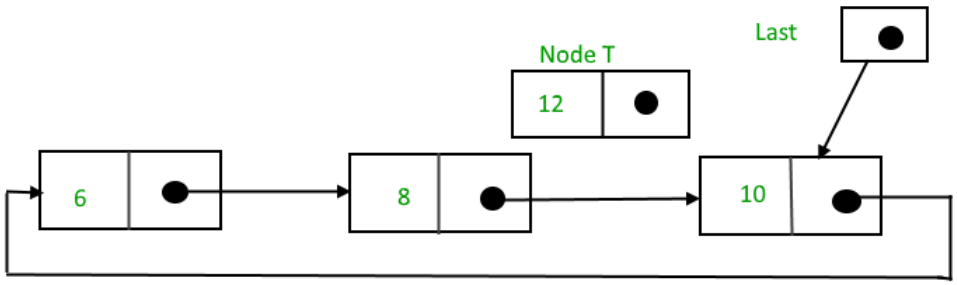

```
struct Node *addEnd(struct Node *last, int data)
{
if (last == NULL)
        return addToEmpty(last, data);

// Creating a node dynamically.
struct Node *temp =
                (struct Node *)malloc(sizeof(struct Node));

// Assigning the data.
temp -> data = data;

// Adjusting the links.
temp -> next = last -> next;
last -> next = temp;
last = temp;

return last;
}
```

Insertion in between the nodes

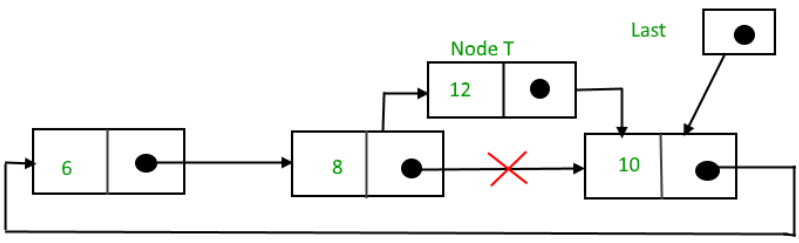
To insert a node in between the two nodes, follow these steps:

1. Create a node, say T.
2. Search for the node after which T needs to be inserted, say that node is P.
3. Make T -> next = P -> next;
4. P -> next = T.

Suppose 12 needs to be inserted after the node has the value 10,



After searching and insertion



Video Content / Details of website for further learning (if any): **https://www.geeksforgeeks.org/circular-singly-linked-list-insertion/** **https://www.geeksforgeeks.org/circular-linked-list/**
Important Books/Journals for further learning including the page nos.: Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :79

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 - 9

AI&DS

Year/Sem

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : I - Introduction **Date of Lecture:**

Topic of Lecture: Application of List: Polynomial Manipulation

Introduction :

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.

Prerequisite knowledge for Complete understanding and learning of Topic:

1.Basics of Algorithm

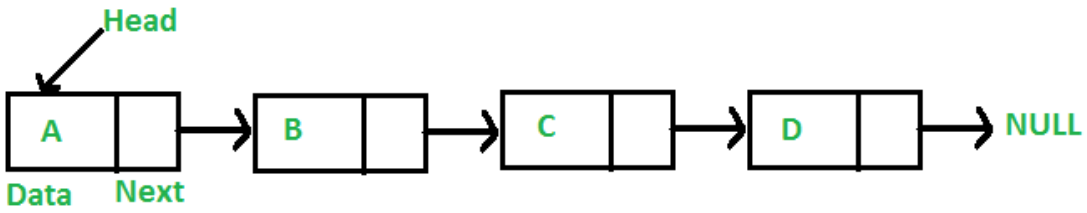
2.Programming Knowledge

3.Data Structure

Detailed content of the Lecture:

Applications of linked list data structure

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



Applications of linked list in computer science –

1. Implementation of stacks and queues
2. Implementation of graphs : Adjacency list representation of graphs is most popular which is uses linked list to store adjacent vertices.
3. Dynamic memory allocation : We use linked list of free blocks.
4. Maintaining directory of names
5. Performing arithmetic operations on long integers
6. Manipulation of polynomials by storing constants in the node of linked list

7. representing sparse matrices

linked list is a data structure that stores each element as an object in a node of the list. every note contains two parts data han and links to the next node.

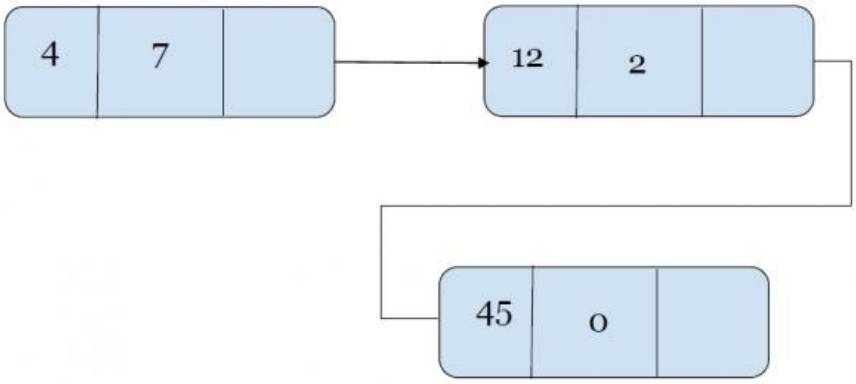
Polynomial is a mathematical expression that consists of variables and coefficients. for example x^2 - 4x + 7

In the **Polynomial linked list**, the coefficients and exponents of the polynomial are defined as the data node of the list.

For adding two polynomials that are stored as a linked list. We need to add the coefficients of variables with the same power. In a linked list node contains 3 members, coefficient value link to the next node.

a linked list that is used to store Polynomial looks like −

Polynomial : $4x^7 + 12x^2 + 45$



This is how a linked list represented polynomial looks like.

Adding two polynomials that are represented by a linked list. We check values at the exponent value of the node. For the same values of exponent, we will add the coefficients.

Example,

Input :

p1= $13x^8 + 7x^5 + 32x^2 + 54$

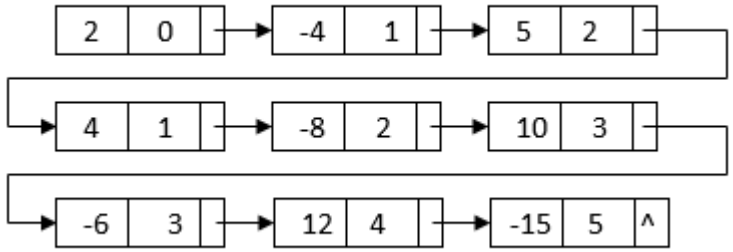
p2= $3x^{12} + 17x^5 + 3x^3 + 98$

Output : $3x^{12} + 13x^8 + 24x^5 + 3x3 + 32x^2 + 152$

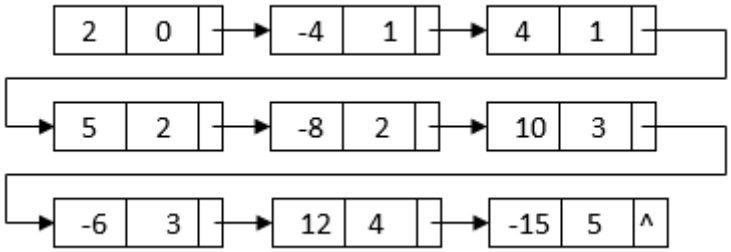
Multiply Two Polynomials

To multiply two polynomials, we can first multiply each term of one polynomial to the other polynomial**.** Suppose the two polynomials have and terms. This process will create a polynomial with terms.

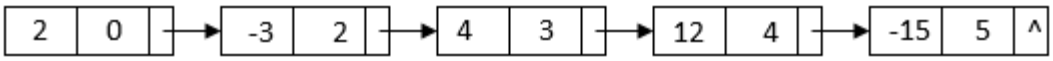
For example, after we multiply and , we can get a linked list:



This linked list contains all terms we need to generate the final result. However, it is not sorted by the powers. Also, it contains duplicate nodes with like terms. **To generate the final linked list, we can first merge sort the linked list based on each node's power:**



After the sorting, the like term nodes are grouped together. **Then, we can merge each group of like terms and get the final multiplication result:**



Video Content / Details of website for further learning (if any):
https://www.baeldung.com/cs/polynomial-multiplication-addition
https://www.tutorialspoint.com/adding-two-polynomials-using-linked-list-in-cplusplus#:~:text=In%20the%20Polynomial%20linked%20list,data%20node%20of%20the%20list.&text=Adding%20two%20polynomials%20that%20are%20represented%20by%20a%20linked%20list,we%20will%20add%20the%20coefficients.

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :79

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-10

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: Stack ADT :Operations

Introduction :

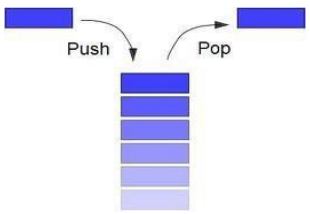
- Stack is an Abstract data structure (ADT) works on the principle Last In First Out (LIFO).
- The last element add to the stack is the first element to be delete. Insertion and deletion can be takes place at one end called TOP. It looks like one side closed tube.

Prerequisite knowledge for Complete understanding and learning of Topic:

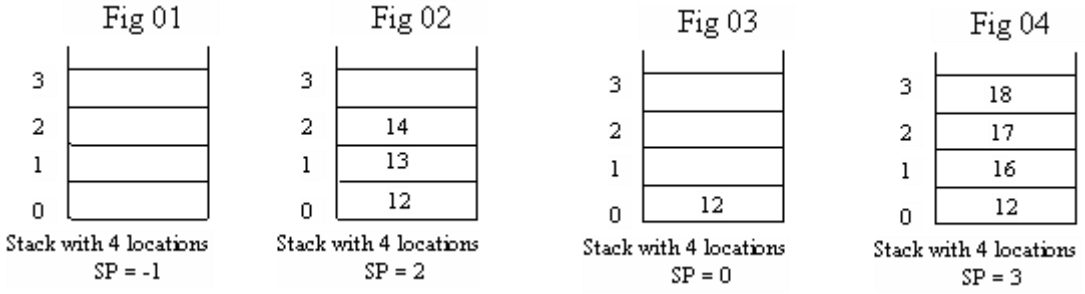
- Abstract data Type
- Arrays
- Linked List

Detailed content of the Lecture:

- A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack.
- A stack is a limited access data structure - elements can be added and removed from the stack only at the top. Push adds an item to the top of the stack, pop removes the item from the top.



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



There are two operations applied on stack they are

- Push
- pop.

Push:

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

```
void push(int data)
    {
            if( ! isFull() )
        {       top = top + 1;
          stack[top] = data; }
            else
        {     cout<<"Could not insert data, Stack is full.\n"
};
}
```

Time Complexity of Push operation in Stack is O(1)

Pop:

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

Pseudocode for pop operation

```
int pop(int data)
 {      if(!isempty())
        {       data = stack[top];
                top = top - 1;
                return data;
         }
        else
         {    cout<<“Stack is empty.\n"; }
}
```

Time Complexity of Pop operation in Stack is O(1)

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

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

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

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

Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :103

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L-11

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: Applications of Stacks: Evaluating Arithmetic Expression

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Stack operation
- Expression

Detailed content of the Lecture:

- Reverse a string
- Check well-formed (nested) parenthesis
- Convert infix expression to postfix expressions
- Evaluate the postfix expression

Expressions are divided into three types

- Infix Expression
- Postfix Expression
- Prefix Expression

Infix Expression
In infix expression,operator is used in between the operands.
A+B
Postfix Expression
In postfix expression, operator is used after operands. that the"Operator follows the Operands".
AB+
Prefix Expression
In prefix expression, operator is used before operands.that the "Operands follows the Operator
+AB

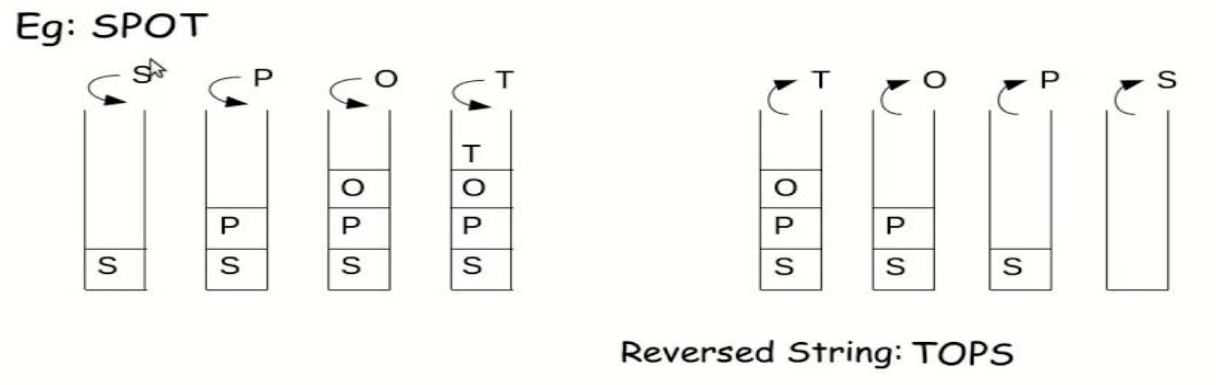
Conversion of infix expression to postfix expression

- Scan the infix expression from left to right.
- If the scanned symbol is left parenthesis, push it onto the stack.
- If the scanned symbol is an operand, then place directly in the postfix expression (output).
- If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.
- If the scanned symbol is an operator, then go on removing all the operators from the stack and

place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (or equal) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

2.Reverse a string

1. Create an empty stack
2. One by one push all characters of string to stack
3 One by one pop all characters from stack and
4. put them back to string.



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

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

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

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

Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :104

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-12

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: Evaluating Arithmetic Expression

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Stack operation
- Expression

Detailed content of the Lecture:

Pseudocode

```
void reverse_string(char str[])
{
        int n = strlen(str);
        struct Stack* stack = createStack(n);
        int i;
            for (i = 0; i < n; i++)
                {    push(stack, str[i]); }
          for (i = 0; i < n; i++)
                {     str[i] = pop(stack); }
}
```

Time complexity to reverse string using stack

= O(1)+ O(1)+ O(n)+ O(1)+ O(n)+ O(1)

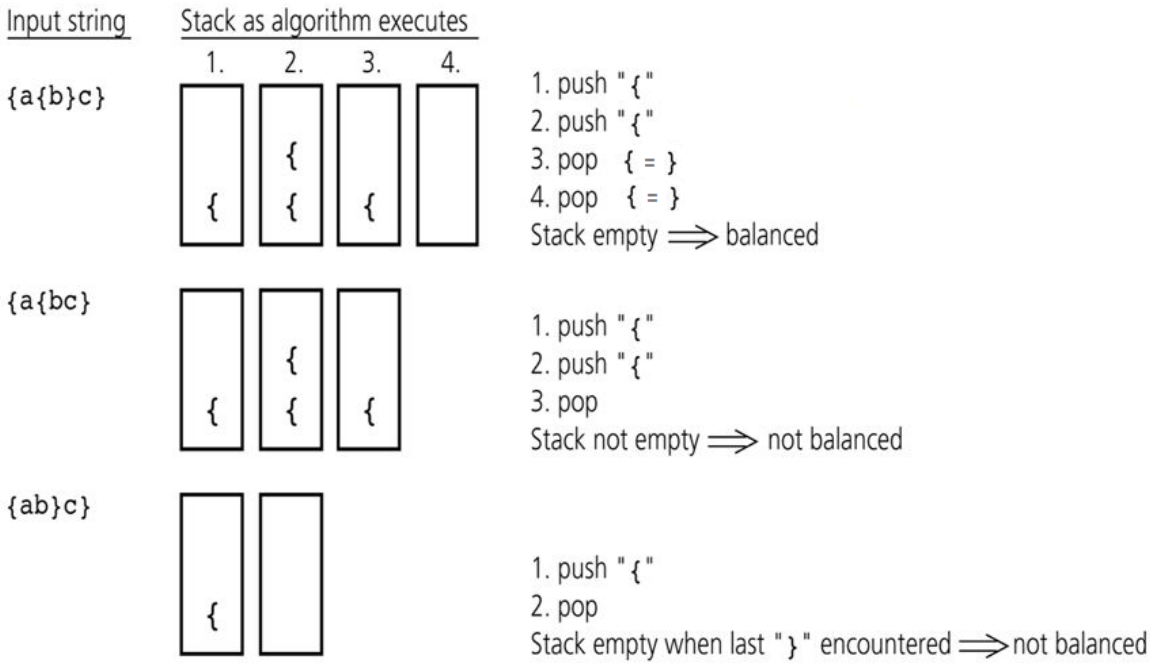
= O(n)+ O(1)

=O(n)

3.Check well-formed (nested) parenthesis / Balanced Parenthesis

Algorithm:

1. Get a String and Empty stack as input
2. Now traverse the string from left to right , and get only the parenthesis of the expression
3. If the current character is a opening bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack
4. If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack
5. And If the popped character is the matching i.e (=) it is balanced else (‡} not balanced
6. Repeat process 2 to 5 until string is end
7. After complete traversal, if there is some open bracket left in stack / no bracket in the stack then also “not balanced” else balanced

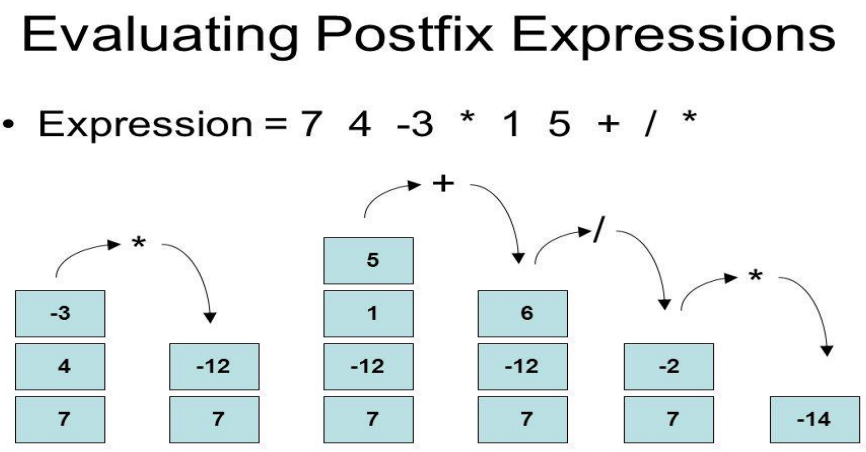


4. Evaluate the postfix expression

Other name of postfix expression is reverse polish notation

Algorithm:

1. Get Postfix Expression and an empty stack as input
2. Scan the postfix expression from left to right
3. If element is an operand, push it into the stack
4. If the element is an operator , pop twice
5. Evaluate expression according to the operator & push the result back to the stack
6. Repeat step 2 to 5 until expression is end
7. The value in the stack is the final answer



Program

```
int evaluatePostfix(char* exp)
{    struct Stack* stack = createStack(strlen(exp));
    for (i = 0; exp[i]; ++i)
 {    if (isdigit(exp[i]))
                push(stack, exp[i] );
                else
                {     int val1 = pop(stack);
                      int val2 = pop(stack);
                             switch (exp[i])
                    {         case '+': push(stack, val2 + val1); break;
                                       case '-': push(stack, val2 - val1); break;
                                       case '*': push(stack, val2 * val1); break;
                                       case '/': push(stack, val2/val1); break;        }
                     }
}   return pop(stack);   }
```

Time complexity

= O(1)+ O(n)+ O(1)
= O(n)+ O(1)
=O(n)

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

https://www.youtube.com/watch?v=sFVxsglODoo
https://nptel.ac.in/courses/106/106/106106133/

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

Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :104

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-13

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: Conversion of Infix to Postfix Expression

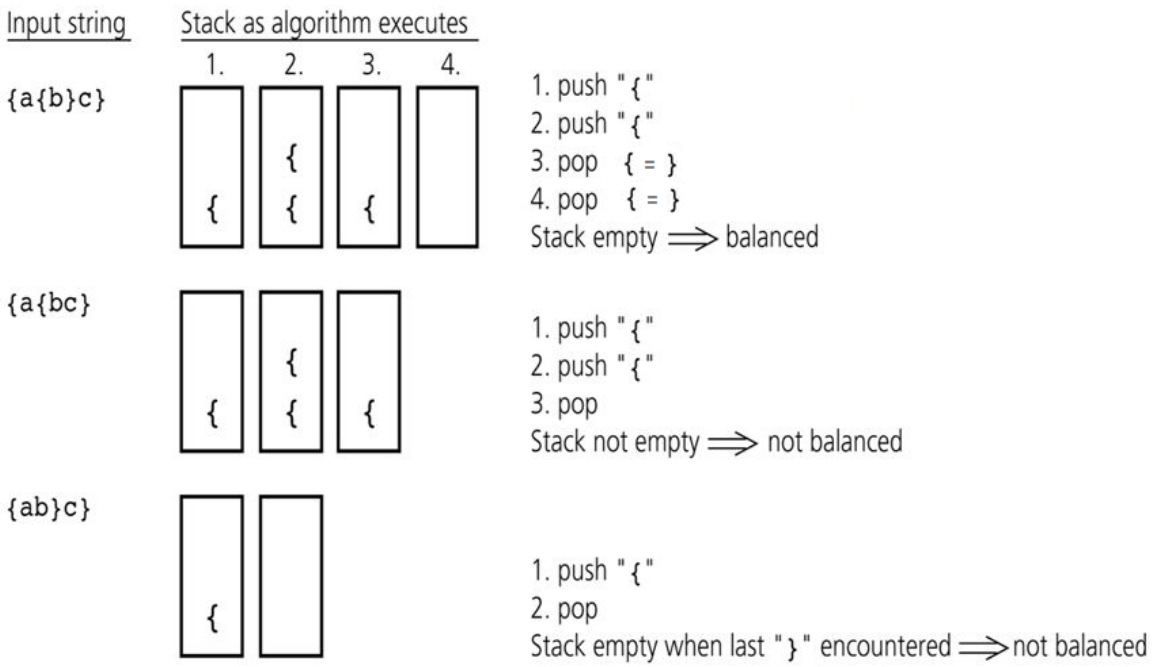
Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Stack operation
- Expression

Detailed content of the Lecture:



Input:

The infix expression. x^y/(5*z)+2

Output:

Postfix Form Is: xy^5z*/2+

Algorithm

infixToPostfix(infix)

Input − Infix expression.

Output − Convert infix expression to postfix form.

```
#include<iostream>
#include<stack>
#include<locale>      //for function isalnum()
using namespace std;

int preced(char ch) {
   if(ch == '+' || ch == '-') {
      return 1;              //Precedence of + or - is 1
   }else if(ch == '*' || ch == '/') {
      return 2;              //Precedence of * or / is 2
   }else if(ch == '^') {
      return 3;              //Precedence of ^ is 3
   }else {
      return 0;
   }
}

string inToPost(string infix ) {
   stack<char> stk;
   stk.push('#');              //add some extra character to avoid underflow
   string postfix = "";        //initially the postfix string is empty
   string::iterator it;

   for(it = infix.begin(); it!=infix.end(); it++) {
      if(isalnum(char(*it)))
         postfix += *it;      //add to postfix when character is letter or number
      else if(*it == '(')
         stk.push('(');
      else if(*it == '^')
         stk.push('^');
      else if(*it == ')') {
         while(stk.top() != '#' && stk.top() != '(') {
            postfix += stk.top(); //store and pop until ( has found
            stk.pop();
         }
         stk.pop();          //remove the '(' from stack
      }else {
         if(preced(*it) > preced(stk.top()))
            stk.push(*it); //push if precedence is high
         else {
            while(stk.top() != '#' && preced(*it) <= preced(stk.top())) {
               postfix += stk.top();       //store and pop until higher precedence is found
               stk.pop();
            }
            stk.push(*it);
         }
      }
   }

   while(stk.top() != '#') {
      postfix += stk.top();       //store and pop until stack is not empty.
```

```
    stk.pop();
  }

  return postfix;
}

int main() {
  string infix = "x^y/(5*z)+2";
  cout << "Postfix Form Is: " << inToPost(infix) << endl;
}
```

Video Content / Details of Itbsite for further learning (if any):
https://www.tutorialspoint.com/Convert-Infix-to-Postfix-Expression
https://runestone.academy/runestone/books/published/pythonds/BasicDS/InfixPrefixandPostfixExpressions.html

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :109

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

L-14

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: QUEUE ADT

Introduction :

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

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

- Data structure
- Indices
- Pointers

Detailed content of the Lecture:

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

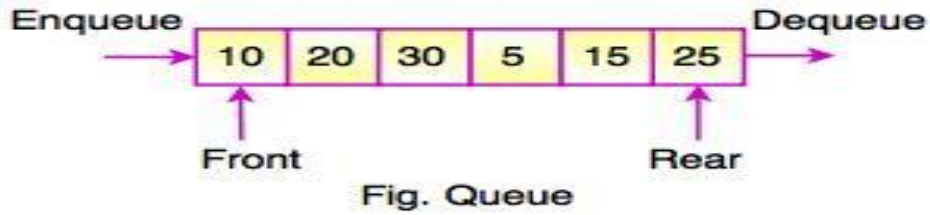


Fig. Queue

- **Enqueue** means to insert an item into the back of the **queue.**
  Step 1 − Check if the queue is empty.
  Step 2 − If the queue is empty, produce underflow error and exit.
  Step 3 − If the queue is not empty, access the data where front is pointing.
  Step 4 − Increment front pointer to point to the next available data element and success.

- **Dequeue** means removing the front item. The picture demonstrates the FIFO access. The difference

between stacks and queues is in removing.

Algorithm :
Step 1 − Check if the queue is full.
Step 2 − If the queue is full, produce overflow error and exit.
Step 3 − If the queue is not full, increment rear pointer to point the next empty space.
Step 4 − Add data element to the queue location, where the rear is pointing and return success.

Operations on Queue

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

Video Content / Details of website for further learning (if any):
https://www.tutorialspoint.com/cplusplus-program-to-implement-queue-using-array#:~:text=A%20queue%20is%20an%20abstract,removed%20first%20in%20a%20queue.
https://www.softwaretestinghelp.com/queue-in-cpp/

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :112

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-15

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: Queue Operations

Introduction :

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

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

- Data structure
- Indices
- Pointers

Detailed content of the Lecture:

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

Types Of Queue

1. Circular Queue
2. Deque (Double Ended Queue)
3. Priority Queue

Enqueue Operation

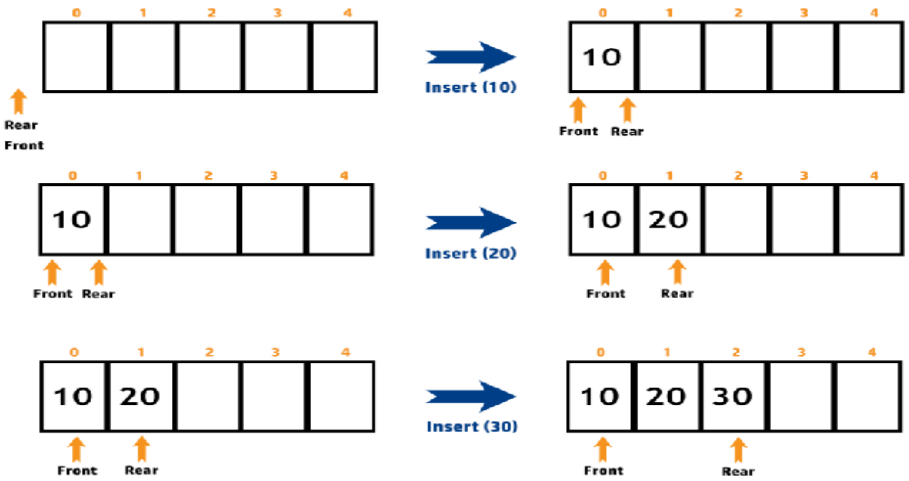
- The process of **adding a new data element** in end of the queue is known as a Enqueue Operation (Rear)
- Enqueue operation involves a series of steps

Step 1 − Checks if the queue is full
Step 2 − If queue **is full, produces an error** and exit
Step 3 − If queue is **not full,**
- If queue **is empty**, **increment both rear and front** pointer by one

- **else increment rear** by one which points to next empty space

Step 4 − **Adds new data** element to the queue , where rear is pointing

Step 5 − Returns success



Disadvantage of Linear Queue

- Now, actually we have **deleted 2** elements from queue
- So, there should be **space for another 2 elements** in the queue
- But as **Rear pointer is pointing at last position** and **Queue overflow** condition
- (Rear == SIZE-1) is true, **we can't insert new element** in the queue even if it has an empty space
- To overcome this problem there is another variation of queue called **circular queue**

Pseudocode for enqueue operation

```
void enqueue(int data)
    {     cout<<"Enter data to insert in a queue\t";
          Cin>>"%d",&data;
              if( ! isFull() )
          {         if(front=-1)
              {    front=front +1; rear=rear+1;
                   queue[rear] = data; }
                          else
              {   rear=rear+1;    queue[rear] = data; }
    }    cout<<"Could not insert data, Queue is full.\n"   ;
}
```

Time Complexity of Enqueue operation in Queue is O(1)

Dequeue Operation

- Removing an element from the queue at front end is known as a Dequeue Operation
- dequeue operation involves a series of steps

Step 1 − Checks if the queue is empty (front==-1)

Step 2 − If the queue is empty, produces an error and exit

Step 3 − else, remove the data element at which front is pointing

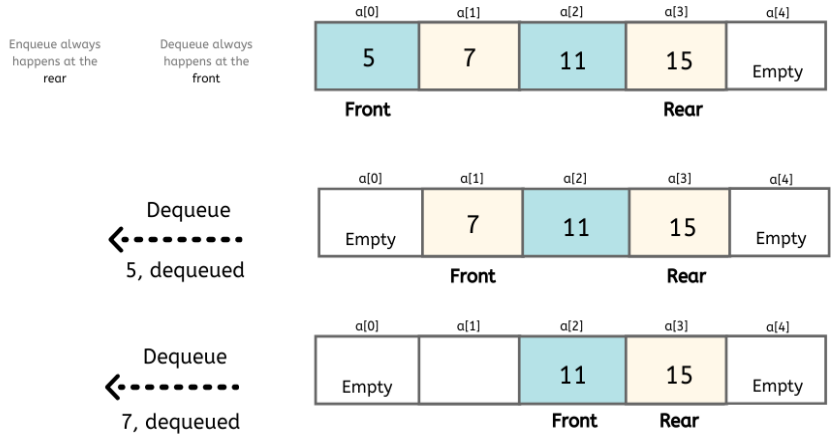
Step 4 − Increment the value of front by 1

Step 5 − Returns success

Pseudocode for dequeue operation

```
int dequeue(int data)
 {       if(! isempty())
         {       data = queue[front];
                 front = front+ 1;
                 return data;
          }
        else
         {     cout<<“queue is empty\n"; }
}
```

Time Complexity of Dequeue operation of Queue is O(1)



Video Content / Details of website for further learning (if any):
https://www.tutorialspoint.com/cplusplus-program-to-implement-queue-using-array#:~:text=A%20queue%20is%20an%20abstract,removed%20first%20in%20a%20queue.
https://www.geeksforgeeks.org/queue-data-structure/

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :112

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-16

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: Circular Queue

Introduction :

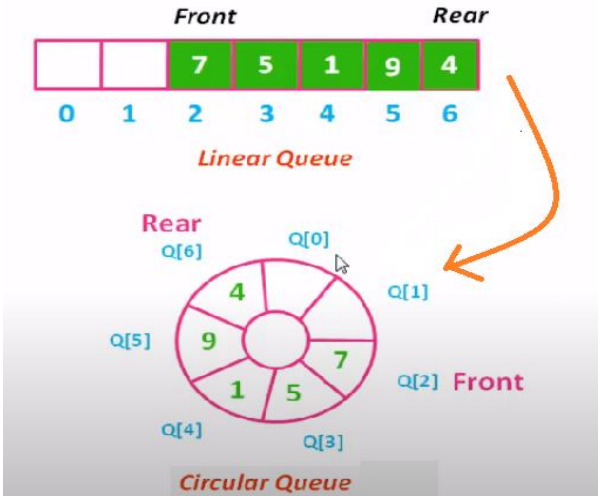
- A queue is a data structure that is best described as "first in, first out".
- A queue is another special kind of list, where items are inserted at one end called the rear and deleted at the other end called the front.
- A real world example of a queue is people waiting in line at the bank. As each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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Abstract data Type
- Data Structures
- Indices

Detailed content of the Lecture:

- Circular Queue is a Linear Data Structure
- Collection of similar data items
- In which enqueue (insertion)operation performed at rear end and
- Dequeue (deletion) operations performed at front end
- It follows FIFO: First In, First Out / LILO: Last In Last Out –principle
- In circular queue the last node is connected back to the first node to make a circle



- Circular Queue is a Linear Data Structure
- Collection of similar data items

- In which enqueue (insertion)operation performed at rear end and
- Dequeue (deletion) operations performed at front end
- It follows FIFO: First In, First Out / LILO: Last In Last Out –principle
- In circular queue the last node is connected back to the first node to make a circle

Basic Operations of Circular Queue (C-Queue ADT)

- **Primary Operations**

- **enqueue()** − Adds an element to the rear of the queue
- **dequeue()** − Removes an element from the front of the queue

- **Secondary Operations**

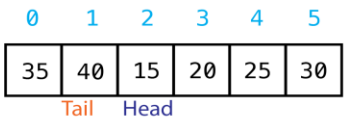
- **peek()** − get the front data element of the queue, without removing it
- **isFull()** − check if queue is full
- **isEmpty()** − check if queue is empty

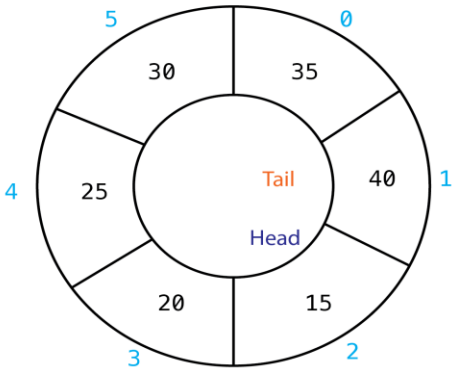
Enqueue Operation

- The process of **adding a new data element at** rear end of the queue is known as a Enqueue Operation
- Enqueue operation involves a series of steps

Step 1 − Checks if the queue is full **(front== (rear+1) % (Maxsize_queue));**
Step 2 − If queue **is full, produces an error** and exit
Step 3 − If queue is **not full,**
- If queue **is empty,(front==-1) increment both rear and front** pointer by one
- **else increment rear** by **(rear = (rear+1) % (Maxsize_queue);**

Step 4 − **Add new data** element to the queue , where rear is pointing
Step 5 − Returns success






```
Q = circularQueue(6)
Q.Enqueue(5)
Q.Enqueue(10)
Q.Enqueue(15)
Q.Enqueue(20)
Q.Enqueue(25)
Q.Enqueue(30)
Q.Dequeue()
Q.Dequeue()
Q.Enqueue(35)
Q.Enqueue(40)
```

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/circular-queue-set-1-introduction-array-implementation/
https://www.tutorialspoint.com/cplusplus-program-to-implement-circular-queue

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :112

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L-17

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: Double Ended Queues

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Abstract data Type
- Data Structures
- Indices

Detailed content of the Lecture:

- Circular Queue is a Linear Data Structure
- Collection of similar data items
- In which enqueue (insertion)operation performed at rear end and
- Dequeue (deletion) operations performed at front end
- It follows FIFO: First In, First Out / LILO: Last In Last Out –principle
- In circular queue the last node is connected back to the first node to make a circle

Dequeue Operation

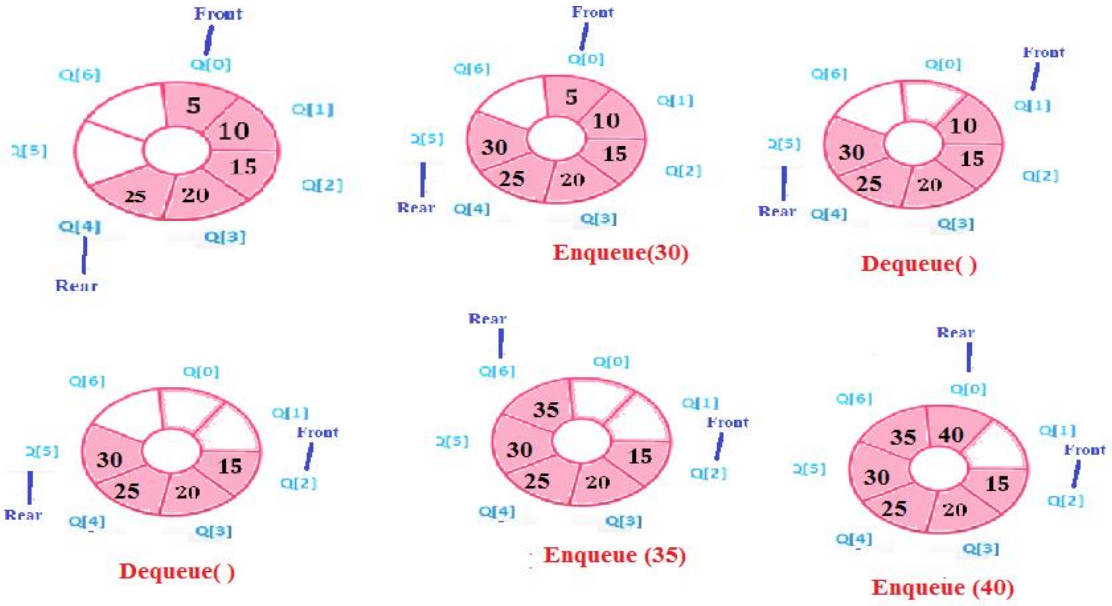
- Removing an element from the queue at front end is known as a Dequeue Operation
- Enqueue operation involves a series of steps

Step 1 − Checks if the queue is empty (**front==-1**)
Step 2 − If the **queue is empty,** print as Queue underflow
Step 3 − **else, check if front=rear ,remove** the data element at which **front** is pointing an **set front=-1 & rear =-1**
Step 4 − **else, remove** the data element at which **front** is pointing and **set front =(front+1)% maxsize_queue**
Step 5 − Returns success

Pseudocode for dequeue operation

```
Circular_dequeue(int data)
 {       if(front==-1)
         {  cout<<"queue is empty" }
           else  { if(front==rear)  data = queue[front];
                front = front- 1 ; rear=rear-1; // increment front by 1
                 return data;    }
           else { data = queue[front];
                          front= =(front+1)% maxsize_queue;
                          return data;    }
  }
```

Time Complexity = O(1)+ O(1)+ O(1)+ O(1)+ O(1)
=O(1)



Video Content / Details of website for further learning (if any):
https://www.wiziq.com/tutorials/data-structure
https://nptel.ac.in/courses/106/106/106106133/

Important Books/Journals for further learning including the page nos.:
Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :112

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-18

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Teacher : M.S.Soundarya

Unit : II – Linear Data Structure **Date of Lecture:**

Topic of Lecture: Application of Queues

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Abstract data Type
- Data Structures
- Indices

Detailed content of the Lecture:

Application of queue*:*

- Implementation of a queue is also an important application of data structure. Nowadays computer handles multiuser, multiprogramming environment and time-sharing environment.
- In this environment a system(computer) handles several jobs at a time, to handle these jobs the concept of a queue is used.
- To implement printer spooler so that jobs can be printed in the order of their arrival.
- Round robin scheduling technique is implemented using queue
- All types of customer service(like railway reservation) centers are designed using the concept of queues.
- Queues find their application in real life within serving requests such as a printer queue, CPU scheduling and calling systems.

Video Content / Details of website for further learning (if any): https://www.codingninjas.com/blog/2020/10/10/queue-data-structure-and-its-applications/ https://techalmirah.com/queue-and-application-of-queue/
Important Books/Journals for further learning including the page nos.: Weiss, Mark Allen. Data structures and algorithm analysis in C++ / Mark Allen Weiss, Florida International University. — Fourth edition 2014 pg-no :112

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

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

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-19

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: Trees - Traversal

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Non-linear data Structure
- Data

Detailed content of the Lecture:

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

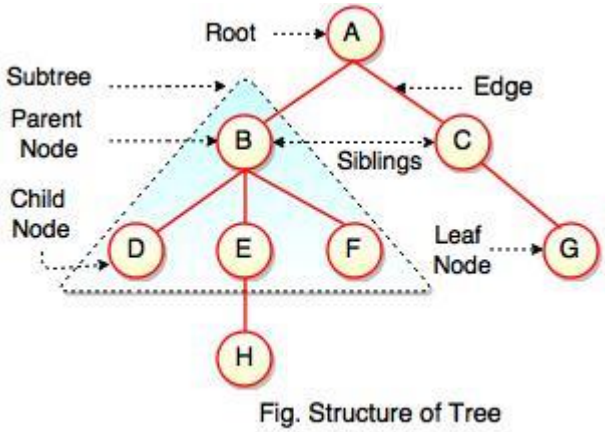


Fig. Structure of Tree

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

Basic terminologies of trees

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

Node

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

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

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

https://www.gatevidyalay.com/tree-data-structure-tree-terminology/
https://www.tutorialride.com/data-structures/trees-in-data-structure.html

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 115-121

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

L-20

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: Binary Tree

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Nodes
- Tree
- Non linear structure

Detailed content of the Lecture:

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

OPERATIONS

(a) Inorder (Left, Root, Right)
(b) Preorder (Root, Left, Right)
(c) Postorder (Left, Right, Root)

Algorithm

1. Traverse the left subtree, i.e., call Inorder (left-subtree)
2. Visit the root
3. Traverse the right subtree, i.e., call Inorder (right-subtree)

Psedocode for Inorder Traversal

```
inOrder(treePointer ptr)
                {
                        if (ptr != NULL)
                        {
                inOrder(ptr->leftChild);
                visit(ptr);
                inOrder(ptr->rightChild);
                        }
                }
```

Preorder Traversal (Root, Left, Right)

Preorder traversal of a binary tree is defined as follow

1. Visit the root
2. Traverse the left subtree, i.e., call Preorder (left-subtree)
3. Traverse the right subtree, i.e., call Preorder (right-subtree)

Pseudocode for Preorder Traversal

```
preOrder (treePointer ptr)
        {
                                        if (ptr != NULL)
                                        {
                                 visit(t);
                                preOrder(ptr->leftChild);
                                preOrder(ptr->rightChild);
                                        }
                        }
```

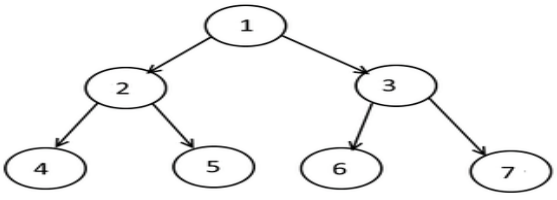
Postorder Traversal (Left, Right, Root)

Postorder traversal of a binary tree is defined as follow

1. Traverse the left subtree, i.e., call Postorder (left-subtree)
2. Traverse the right subtree, i.e., call Postorder (right subtree)
3. Visit the root

Psedocode for Postorder Traversal

```
postOrder(treePointer ptr)
                        {
                                        if (ptr != NULL)
                                        {
                                postOrder(ptr->leftChild);
                                postOrder(ptr->rightChild);
                                visit(t);
                                        }
                        }
```



Inorder Traversal: 4 2 5 1 6 3 7
Preorder Traversal: 1 2 4 5 3 6 7
Postorder Traversal: 7 6 3 5 4 2 1
Breadth-First Search: 1 2 3 4 5 6 7
Depth-First Search: 1 2 4 5 3 6 7

Video Content / Details of website for further learning (if any): https://www.youtube.com/watch?v=sFVxsglODoo https://nptel.ac.in/courses/106/106/106106133/
Important Books/Journals for further learning including the page nos.: Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 115-121

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

L-21

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: Expression Tree

Introduction :

A binary expression tree is a specific kind of a binary tree used to represent expressions. Two common types of expressions that a binary expression tree can represent are algebraic and boolean. These trees can represent expressions that contain both unary and binary operators.

Prerequisite knowledge for Complete understanding and learning of Topic:

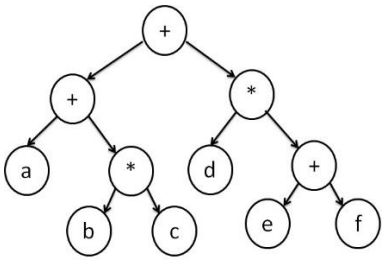
- Nodes
- Tree
- Non linear structure

Detailed content of the Lecture:

Expression Tree is used to represent expressions.

An expression and expression tree shown below

a + (b * c) + d * (e + f)



Expression Tree

All the below are also expressions. Expressions may includes constants value as well as variables

a * 6

16

(a^2)+(b^2)+(2 * a * b)

(a/b) + (c)

m * (c ^ 2)

It is quite common to use parenthesis in order to ensure correct evaluation of expression as shown above
There are different types of expression formats:

- Prefix expression
- Infix expression and
- Postfix expression

Expression Tree is a special kind of binary tree with the following properties:

- Each leaf is an operand. Examples: a, b, c, 6, 100
- The root and internal nodes are operators. Examples: +, -, *, /, ^
- Subtrees are subexpressions with the root being an operator.

Inorder Traversal

We can produce an infix expression by recursively printing out

- the left expression,
- the root, and
- the right expression.

Postorder Traversal

The postfix expression can be evaluated by recursively printing out

- the left expression,
- the right expression and
- then the root

Preorder Traversal

We can also evaluate prefix expression by recursively printing out:

- the root,
- the left expressoion and
- the right expression.

If we apply all these strategies to the sample tree above, the outputs are:

- Infix expression:

```
(a+(b*c))+(d*(e + f))
```

- Postfix Expression:

```
a b c * + d e f + * +
```

- Prefix Expression:

```
+ + a * b c * d + e f
```

Construction of Expression Tree

Let us consider a **postfix expression** is given as an input for constructing an expression tree. Following are the step to construct an expression tree:

1. Read one symbol at a time from the postfix expression.
2. Check if the symbol is an operand or operator.
3. If the symbol is an operand, create a one node tree and push a pointer onto a stack
4. If the symbol is an operator, pop two pointers from the stack namely T_1 & T_2 and form a new tree with root as the operator, T_1 & T_2 as a left and right child
5. A pointer to this new tree is pushed onto the stack

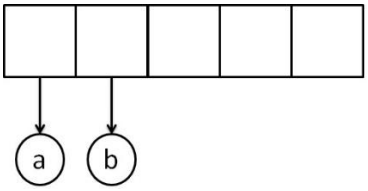
Thus, An expression is created or constructed by reading the symbols or numbers from the left. If operand, create a node. If operator, create a tree with operator as root and two pointers to left and right subtree.

Example - Postfix Expression Construction

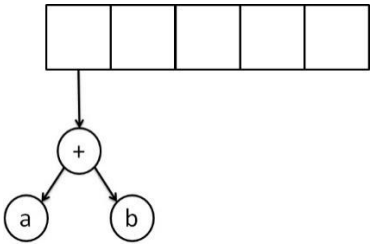
The input is:

`a b + c *`

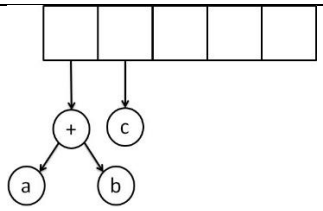
The first two symbols are operands, we create one-node tree and push a pointer to them onto the stack.



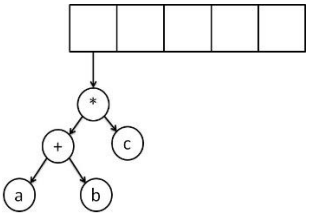
Next, read a'+' symbol, so two pointers to tree are popped,a new tree is formed and push a pointer to it onto the stack.



Next, '**c**' is read, we create one node tree and push a pointer to it onto the stack.



Finally, the last symbol is read ' * ', we pop two tree pointers and form a new tree with a, ' * ' as root, and a pointer to the final tree remains on the stack.



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

https://www.krivalar.com/data-structures-expression-tree
https://www.youtube.com/watch?v=_LxbhLNRZkI

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 115-121

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

L-22

LECTURE HANDOUTS

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: Application of Trees

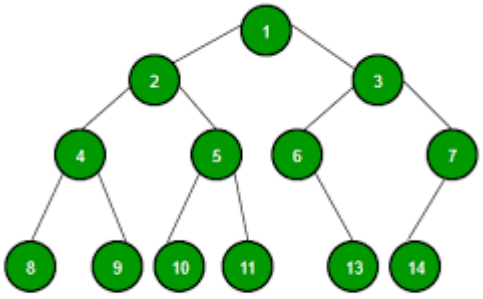
Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- Nodes
- Tree
- Non linear structure

Detailed content of the Lecture:



Why Tree?

Unlike Array and Linked List, which are linear data structures, tree is hierarchical (or non-linear) data structure.

1. 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

2. 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.
3. 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.
4. 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 :

1. Store hierarchical data, like folder structure, organization structure, XML/HTML data.
2. Binary Search Tree is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item
3. Heap is a tree data structure which is implemented using arrays and used to implement priority queues.
4. B-Tree and B+ Tree : They are used to implement indexing in databases.
5. Syntax Tree: Used in Compilers.
6. K-D Tree: A space partitioning tree used to organize points in K dimensional space.
7. Trie : Used to implement dictionaries with prefix lookup.
8. Suffix Tree : For quick pattern searching in a fixed text.
9. Spanning Trees and shortest path trees are used in routers and bridges respectively in computer networks
10. As a workflow for compositing digital images for visual effects.

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

https://www.geeksforgeeks.org/applications-of-tree-data-structure/
https://www.youtube.com/watch?v=URRNruf2yVk

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 121-123

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-23

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: Binary Search Tree

Introduction :

- The nodes of a binary tree can be numbered in a natural way, level by level, left to right.

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

Prerequisite knowledge for Complete understanding and learning of Topic:

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

Detailed content of the Lecture:

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

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

Only smaller values in its left sub tree

Only larger values in its right sub tree

Binary Search Tree Construction-

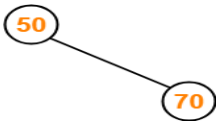
Construct a Binary Search Tree (BST) for the following sequence of numbers-

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

Insert **50**-

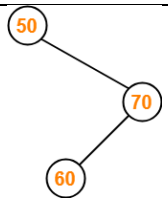
Insert **70**-

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



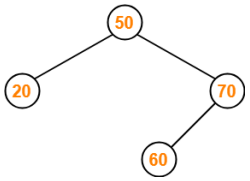
Insert **60**-

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



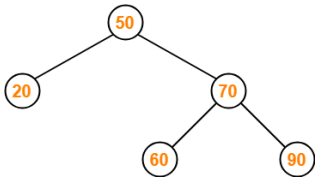
Insert **20**-

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



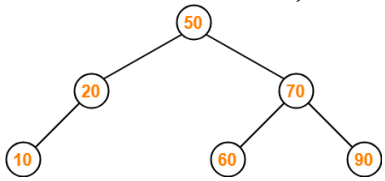
Insert **90**-

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



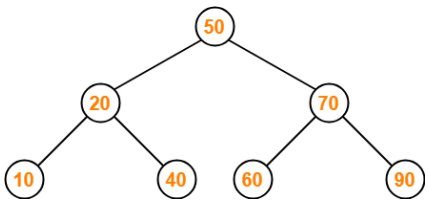
Insert **10**-

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



Insert **40**-

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



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

Searching a key

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

```
# A utility function to search a given key in
BST def search(root,key):
# Base Cases: root is null or key is present at root
    if root is None or root.val ==
```

```
    key: return root
  # Key is greater than root's
 key if root.val < key:
    return search(root.right,key)
  # Key is smaller than root's
key return search(root.left,key)
```

APPLICATIONS OF BINARY SEARCH TREE

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

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

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

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 130-132

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-24

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: AVL Tree

Introduction :

AVL tree is a binary search tree in which the difference of heights of left and right subtrees of any node is less than or equal to one. The technique of balancing the height of binary trees was developed by Adelson, Velskii, and Landi and hence given the short form as AVL tree or Balanced Binary Tree.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Nodes
- Tree
- Non linear structure

Detailed content of the Lecture:

An AVL tree can be defined as follows:

Let T be a non-empty binary tree with T_L and T_R as its left and right subtrees. The tree is height balanced if:

T_L and T_R are height balanced

$h_L - h_R <= 1$, where $h_L - h_R$ are the heights of T_L and T_R

The Balance factor of a node in a binary tree can have value 1, -1, 0, depending on whether the height of its left subtree is greater, less than or equal to the height of the right subtree.

Advantages of AVL tree

Since AVL trees are height balance trees, operations like insertion and deletion have low time complexity. Let us consider an example:

If you have the following tree having keys 1, 2, 3, 4, 5, 6, 7 and then the binary tree will be like the second figure:

Insertion of Trees

Step 1: First, insert a new element into the tree using BST's (Binary Search Tree) insertion logic.
Step 2: After inserting the elements you have to check the Balance Factor of each node.
Step 3: When the Balance Factor of every node will be found like 0 or 1 or -1 then the algorithm will proceed for the next operation.
Step 4: When the balance factor of any node comes other than the above three values then the tree is said to be imbalanced. Then perform the suitable Rotation to make it balanced and then the algorithm will proceed for the next operation.

For Deletion:

Step 1: Firstly, find that node where k is stored
Step 2: Secondly delete those contents of the node (Suppose the node is x)
Step 3: Claim: Deleting a node in an AVL tree can be reduced by deleting a leaf. There are three possible cases:
Step 3: Claim: Deleting a node in an AVL tree can be reduced by deleting a leaf. There are three possible cases:

When x has no children then, delete x
When x has one child, let x' becomes the child of x.
x' cannot have a child, since subtrees of T can differ in height by at most one :

- then replace the contents of x with the contents of x'
- then delete x' (a leaf)

Step 4: When x has two children,

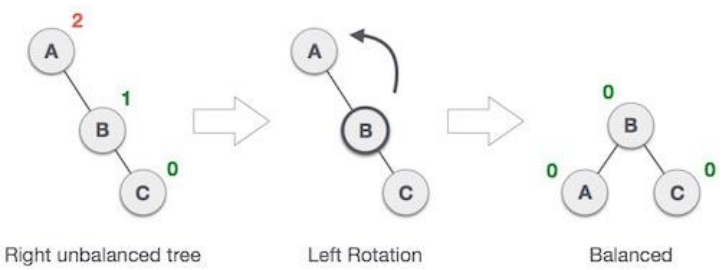
- then find x's successor z (which has no left child)
- then replace x's contents with z's contents, and
- delete z

AVL Rotations

- Left rotation
- Right rotation
- Left-Right rotation
- Right-Left rotation

Left Rotation

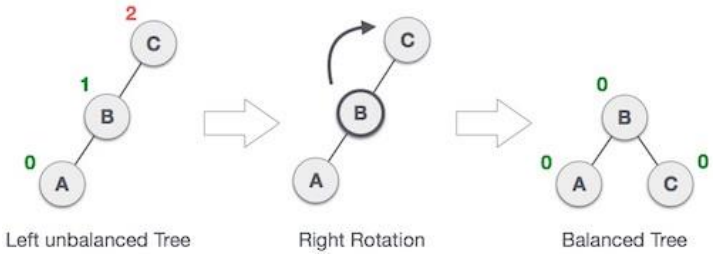
If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation −



In our example, node **A** has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making **A** the left-subtree of B.

Right Rotation

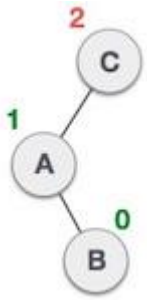
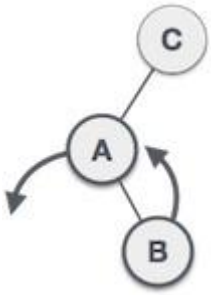
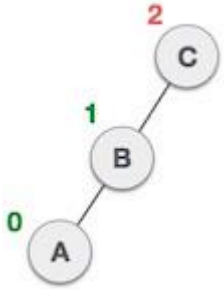
AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.

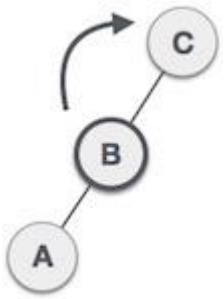
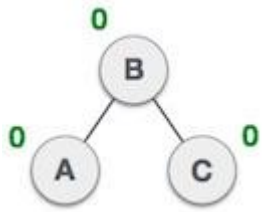


As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

Left-Right Rotation

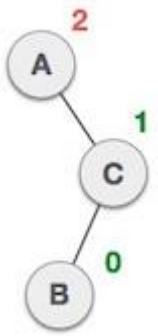
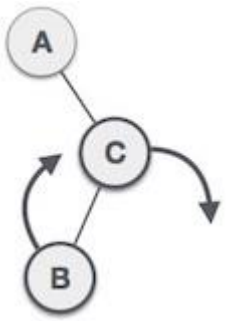
Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

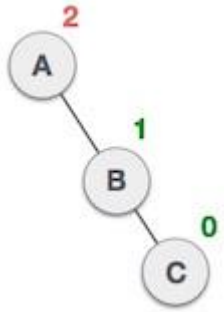
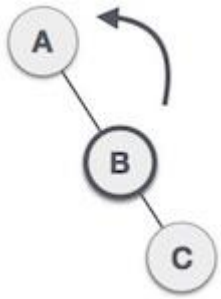
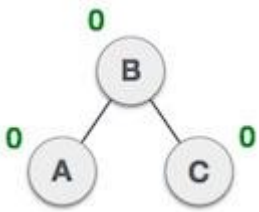
State	Action
	A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**.
	Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree.

	We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree.
	The tree is now balanced.

Right-Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

State	Action
	A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2.
	First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**.

	Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation.
	A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**.
	The tree is now balanced.

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

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 144

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-25

AI&DS

II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: B-Tree

Introduction :

B Tree is a specialized m-way tree that can be widely used for disk access. A B-Tree of order m can have at most m-1 keys and m children. One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.

Prerequisite knowledge for Complete understanding and learning of Topic:

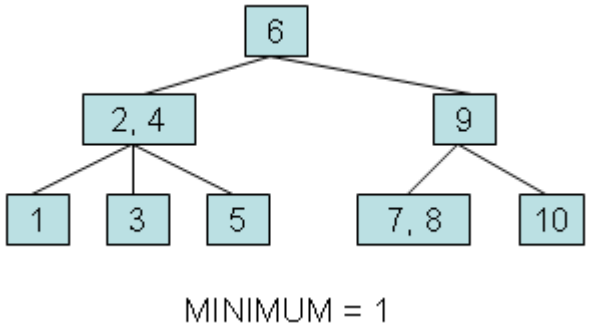
- Nodes
- Tree
- Non linear structure

Detailed content of the Lecture:

B Tree is a specialized m-way tree that can be widely used for disk access. A B-Tree of order m can have at most m-1 keys and m children. One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.

A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

1. Every node in a B-Tree contains at most m children.
2. Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
3. The root nodes must have at least 2 nodes.
4. All leaf nodes must be at the same level



MINIMUM = 1

Operations

Searching :

Searching in B Trees is similar to that in Binary search tree. For example, if we search for an item 49 in the following B Tree. The process will something like following :

1. Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
2. Since, 40<49<56, traverse right sub-tree of 40.
3. 49>45, move to right. Compare 49.
4. match found, return.

Inserting

Insertions are done at the leaf node level. The following algorithm needs to be followed in order to insert an item into B Tree.

1. Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
2. If the leaf node contain less than m-1 keys then insert the element in the increasing order.
3. Else, if the leaf node contains m-1 keys, then follow the following steps.
    - Insert the new element in the increasing order of elements.
    - Split the node into the two nodes at the median.
    - Push the median element upto its parent node.

Deletion

Deletion is also performed at the leaf nodes. The node which is to be deleted can either be a leaf node or an internal node. Following algorithm needs to be followed in order to delete a node from a B tree.

1. Locate the leaf node.
2. If there are more than m/2 keys in the leaf node then delete the desired key from the node.
3. If the leaf node doesn't contain m/2 keys then complete the keys by taking the element from eight or left sibling.
    - If the left sibling contains more than m/2 elements then push its largest element up to its parent and move the intervening element down to the node where the key is deleted.
    - If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.
4. If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.

5. If parent is left with less than m/2 nodes then, apply the above process on the parent too.

If the the node which is to be deleted is an internal node, then replace the node with its in-order successor or predecessor. Since, successor or predecessor will always be on the leaf node hence, the process will be similar as the node is being deleted from the leaf node.

Application of B tree

B tree is used to index the data and provides fast access to the actual data stored on the disks since, the access to value stored in a large database that is stored on a disk is a very time consuming process.

Searching an un-indexed and unsorted database containing n key values needs O(n) running time in worst case. However, if we use B Tree to index this database, it will be searched in O(log n) time in worst case.

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

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 168

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

L-26

AI&DS	LECTURE HANDOUTS	II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: Heap

Introduction :

A Heap is a special Tree-based data structure in which the tree is a complete binary tree.

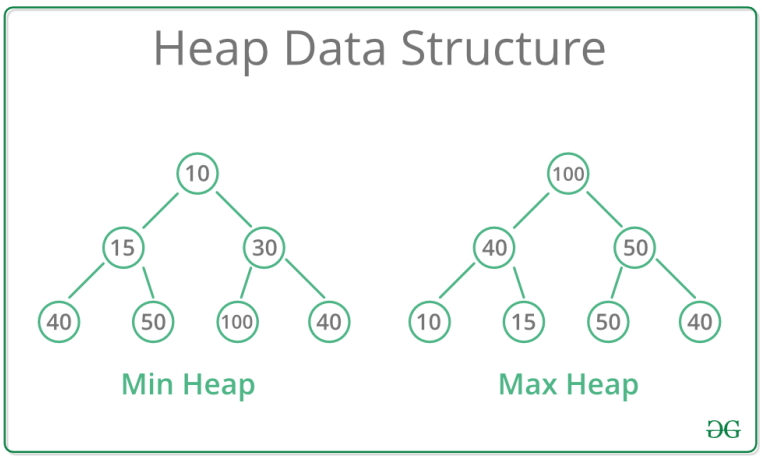
Prerequisite knowledge for Complete understanding and learning of Topic:

- Nodes
- Tree
- Non linear structure

Detailed content of the Lecture:

Heaps can be of two types:

1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.



Binary Heap

A Binary Heap is a Binary Tree with following properties.

1) It's a complete tree (All levels are completely filled except possibly the last level and the last level

has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

Examples of Min Heap:

```
        10                  10
      /    \              /    \
     20     100          15     30
    /                  /  \    /  \
   30                 40   50  100  40
```

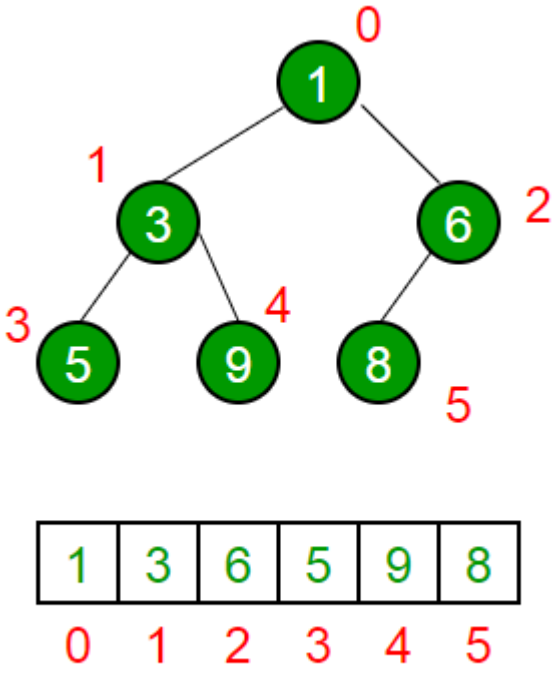
How is Binary Heap represented?

A Binary Heap is a Complete Binary Tree. A binary heap is typically represented as an array.

- The root element will be at Arr[0].
- Below table shows indexes of other nodes for the i^{th} node, i.e., Arr[i]:

Arr[(i-1)/2]	Returns the parent node
Arr[(2*i)+1]	Returns the left child node
Arr[(2*i)+2]	Returns the right child node

- The traversal method use to achieve Array representation is **Level Order**



Operations on Min Heap:

1) getMini(): It returns the root element of Min Heap. Time Complexity of this operation is O(1).
2) extractMin(): Removes the minimum element from MinHeap. Time Complexity of this Operation is O(Logn) as this operation needs to maintain the heap property (by calling heapify()) after removing root.

3) decreaseKey(): Decreases value of key. The time complexity of this operation is O(Logn). If the decreases key value of a node is greater than the parent of the node, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.
Video Content / Details of website for further learning (if any): **https://www.geeksforgeeks.org/time-complexity-of-building-a-heap/** **https://www.youtube.com/watch?v=86mQ1gD3Zgg**
Important Books/Journals for further learning including the page nos.: Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 245

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-27

AI&DS	LECTURE HANDOUTS	II/III

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : III – Non Linear Data Structure **Date of Lecture:**

Topic of Lecture: Application of Heap-Tries

Introduction :

Heaps are used in many famous algorithms such as Dijkstra's algorithm for finding the shortest path, the heap sort sorting algorithm, implementing priority queues, and more. Essentially, heaps are the data structure you want to use when you want to be able to access the maximum or minimum element very quickly.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Nodes
- Tree
- Non linear structure

Detailed content of the Lecture:

Heap Data Structure is generally taught with Heapsort. Heapsort algorithm has limited uses because Quicksort is better in practice. Nevertheless, the Heap data structure itself is enormously used. Following are some uses other than Heapsort.

Priority Queues: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in O(logn) time. Binomoial Heap and Fibonacci Heap are variations of Binary Heap. These variations perform union also in O(logn) time which is a O(n) operation in Binary Heap. Heap Implemented priority queues are used in Graph algorithms like Prim's Algorithm and Dijkstra's algorithm.

The heap data structure has many applications.

- **Heapsort:** One of the best sorting methods being in-place and with no quadratic worst-case scenarios.
- **Selection algorithms**: A heap allows access to the min or max element in constant time, and other selections (such as median or kth-element) can be done in sub-linear time on data that is in a heap.[19]
- **Graph algorithms**: By using heaps as internal traversal data structures, run time will be reduced by polynomial order. Examples of such problems are Prim's minimal-spanning-tree algorithm and Dijkstra's shortest-path algorithm.
- **Priority Queue:** A priority queue is an abstract concept like "a list" or "a map"; just as a list can be implemented with a linked list or an array, a priority queue can be implemented with a heap or a variety of other methods.

- **K-way merge**: A heap data structure is useful to merge many already-sorted input streams into a single sorted output stream. Examples of the need for merging include external sorting and streaming results from distributed data such as a log structured merge tree. The inner loop is obtaining the min element, replacing with the next element for the corresponding input stream, then doing a sift-down heap operation. (Alternatively the replace function.) (Using extract-max and insert functions of a priority queue are much less efficient.)
- **Order statistics:** The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array.

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

https://en.wikipedia.org/wiki/Heap_%28data_structure%29#Applications
https://www.geeksforgeeks.org/applications-of-heap-data-structure/

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 245

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

L - 28

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

Topic of Lecture: Graph – Representation of Graph

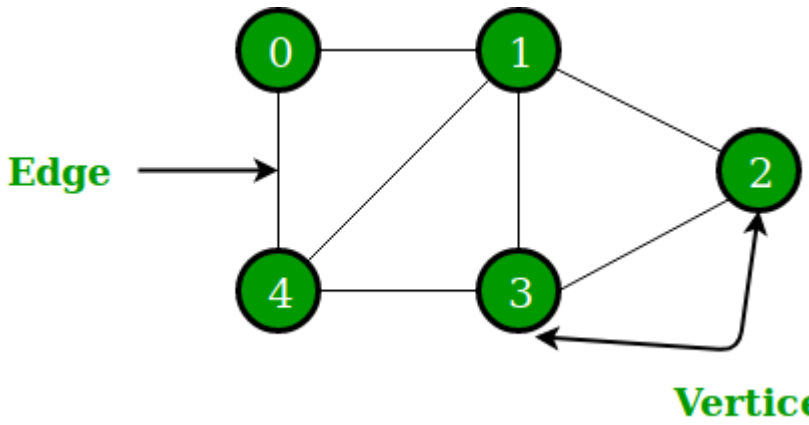
Introduction :

- A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.

Prerequisite knowledge for Complete understanding and learning of Topic:

- **Node**
- **Edge**
- **Vertices**

Detailed content of the Lecture:



In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.

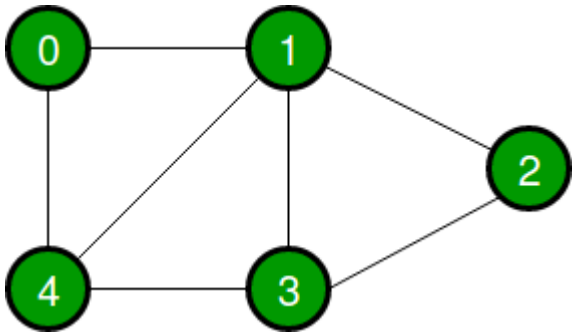
Graph and its representations

A graph is a data structure that consists of the following two components:

1. A finite set of vertices also called as nodes.

2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not

the same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost. Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, and locale. See this for more applications of graph.
Following is an example of an undirected graph with 5 vertices.



The following two are the most commonly used representations of a graph.
1. Adjacency Matrix
2. Adjacency List
There are other representations also like, Incidence Matrix and Incidence List. The choice of graph representation is situation-specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:

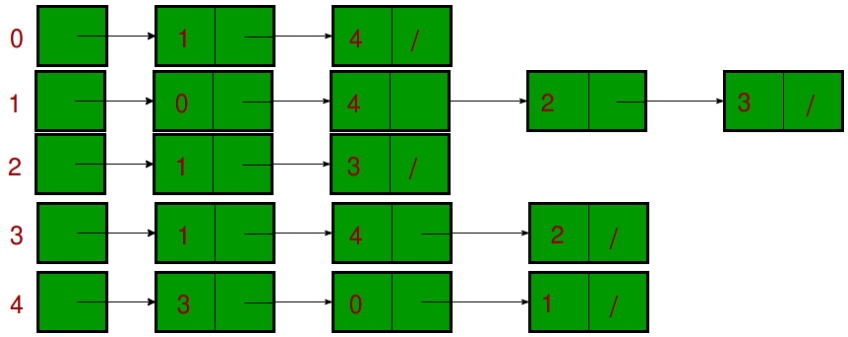
	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).
Cons: Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.
Please see this for a sample Python implementation of adjacency matrix.

Adjacency List:

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

pairs. Following is the adjacency list representation of the above graph.



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

https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
https://www.geeksforgeeks.org/graph-and-its-representations/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 29

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

Topic of Lecture: Graph Traversal

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- **Node**
- **Edge**
- **Vertices**

Detailed content of the Lecture:

Graph traversal is a technique used for a searching vertex in a graph. The graph traversal is also used to decide the order of vertices is visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

There are two graph traversal techniques and they are as follows...

1. **DFS (Depth First Search)**
2. **BFS (Breadth First Search)**

DFS (Depth First Search)

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

DFS (Depth First Search)

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

Video Content / Details of website for further learning (if any): **http://www.btechsmartclass.com/data_structures/graph-traversal-dfs.html** **http://www.btechsmartclass.com/data_structures/graph-traversal-bfs.html**
Important Books/Journals for further learning including the page nos.: Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 30

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

Topic of Lecture: Breadth First Traversal

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- **Node**
- **Edge**
- **Vertices**

Detailed content of the Lecture:

The following steps to implement BFS traversal...

- **Step 1 -** Define a Queue of size total number of vertices in the graph.
- **Step 2 -** Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.
- **Step 3 -** Visit all the non-visited **adjacent** vertices of the vertex which is at front of the Queue and insert them into the Queue.
- **Step 4 -** When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
- **Step 5 -** Repeat steps 3 and 4 until queue becomes empty.
- **Step 6 -** When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, a node may be visited twice. To avoid processing a node more than once, use a boolean visited array.

Example:

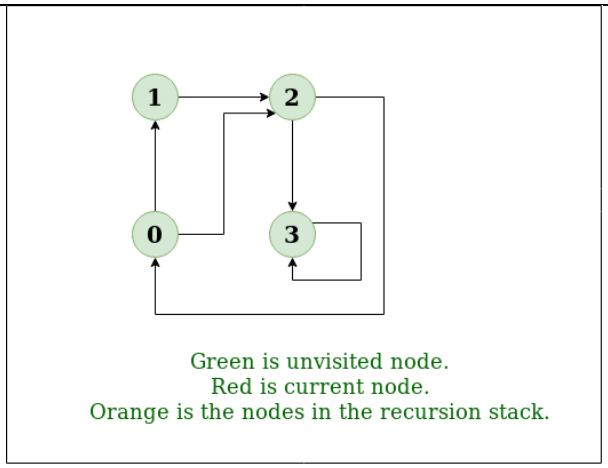
Input: *n = 4, e = 6*

0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

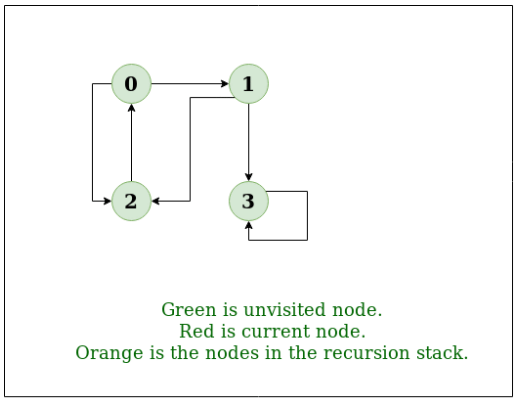
Output: *DFS from vertex 1 : 1 2 0 3*

Explanation:

DFS Diagram:



Input: *n = 4, e = 6*
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3
Output: *DFS from vertex 2 : 2 0 1 3*
Explanation:
DFS Diagram:




```
// C++ program to print DFS traversal from
// a given vertex in a given graph
#include <bits/stdc++.h>
using namespace std;

// Graph class represents a directed graph
// using adjacency list representation
class Graph
{
public:
        map<int, bool> visited;
        map<int, list<int>> adj;

        // function to add an edge to graph
        void addEdge(int v, int w);

        // DFS traversal of the vertices
        // reachable from v
        void DFS(int v);
};

void Graph::addEdge(int v, int w)
{
        adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFS(int v)
```

```
{
        // Mark the current node as visited and
        // print it
        visited[v] = true;
        cout << v << " ";

        // Recur for all the vertices adjacent
        // to this vertex
        list<int>::iterator i;
        for (i = adj[v].begin(); i != adj[v].end(); ++i)
                if (!visited[*i])
                        DFS(*i);
}

// Driver code
int main()
{
        // Create a graph given in the above diagram
        Graph g;
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        cout << "Following is Depth First Traversal"
                        " (starting from vertex 2) \n";
        g.DFS(2);

        return 0;
}
```

Output:

Following is Depth First Traversal (starting from vertex 2)

2 0 1 3

Complexity Analysis:

- **Time complexity:** O(V + E), where V is the number of vertices and E is the number of edges in the graph.
- **Space Complexity:** O(V).
  Since an extra visited array is needed of size V.

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

https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
https://www.google.com/search?q=graph+in+data+structure&sxsrf=AOaemvKNSfdSUMD

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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

L - 31

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

Topic of Lecture: Depth First Traversal

Introduction :

- Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, a node may be visited twice. To avoid processing a node more than once, use a boolean visited array.

Prerequisite knowledge for Complete understanding and learning of Topic:

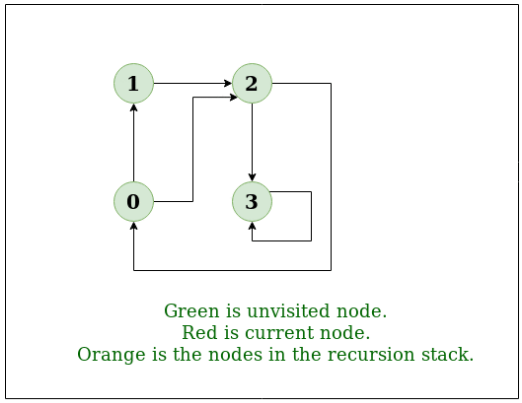
- **Node**
- **Edge**
- **Vertices**

Detailed content of the Lecture:

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, a node may be visited twice. To avoid processing a node more than once, use a boolean visited array.

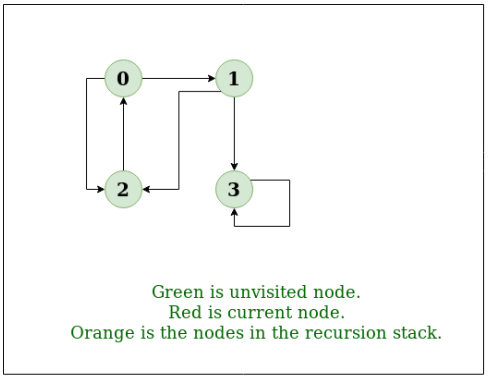
Example:

Input: *n = 4, e = 6*
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3
Output: *DFS from vertex 1 : 1 2 0 3*
Explanation:
DFS Diagram:



Input: *n = 4, e = 6*
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Output: *DFS from vertex 2 : 2 0 1 3*
Explanation:
DFS Diagram:



- **Approach:** Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.
- **Algorithm:**
  1. Create a recursive function that takes the index of the node and a visited array.
  2. Mark the current node as visited and print the node.
  3. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
https://www.youtube.com/watch?v=UxXasZyx0Z0

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 32

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

Topic of Lecture: Minimum Spanning Trees Prim's algorithm

Introduction :

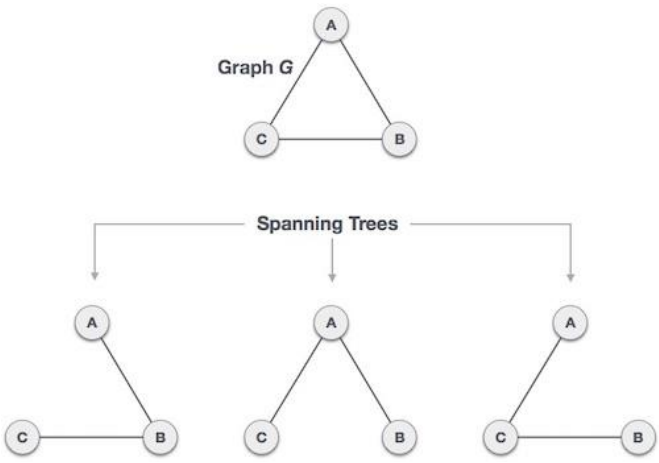
- Graph is a non linear data structure that has nodes and edges. Minimum Spanning Tree is **a set of edges in an undirected weighted graph** that connects all the vertices with no cycles and minimum total edge weight. When number of edges to vertices is high, Prim's algorithm is preferred over Kruskal's.

Prerequisite knowledge for Complete understanding and learning of Topic:

- **Node**
- **Edge**
- **Vertices**

Detailed content of the Lecture:

- A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



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

General Properties of Spanning Tree

We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph G −

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is **minimally connected**.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is **maximally acyclic**.

Mathematical Properties of Spanning Tree

- Spanning tree has **n-1** edges, where **n** is the number of nodes (vertices).
- From a complete graph, by removing maximum **e - n + 1** edges, we can construct a spanning tree.
- A complete graph can have maximum $\mathbf{n^{n-2}}$ number of spanning trees.

Thus, we can conclude that spanning trees are a subset of connected Graph G and disconnected graphs do not have spanning tree.

Application of Spanning Tree

Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common application of spanning trees are −

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

Let us understand this through a small example. Consider, city network as a huge graph and now plans to deploy telephone lines in such a way that in minimum lines we can connect to all city nodes. This is where the spanning tree comes into picture.

Minimum Spanning Tree (MST)

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

Minimum Spanning-Tree Algorithm

We shall learn about two most important spanning tree algorithms here −

- Kruskal's Algorithm
- Prim's Algorithm

Video Content / Details of website for further learning (if any):
https://www.tutorialspoint.com/data_structures_algorithms/spanning_tree.html

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 33

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

Topic of Lecture: Kruskal's algorithm

Introduction :

- Kruskal's algorithm finds **a minimum spanning forest of an undirected edge-weighted graph**. If the graph is connected, it finds a minimum spanning tree. ... It is a greedy algorithm in graph theory as in each step it adds the next lowest-weight edge that will not form a cycle to the minimum spanning forest.

Prerequisite knowledge for Complete understanding and learning of Topic:

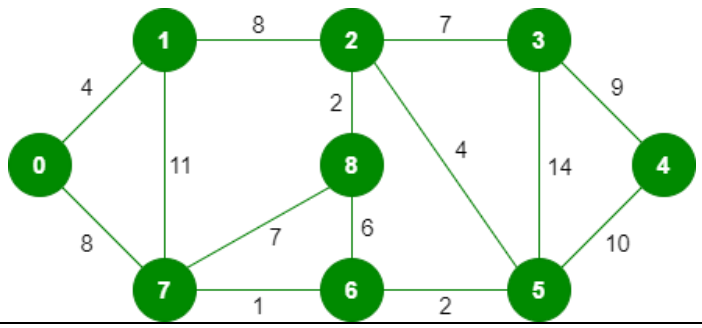
- **Node**
- **Edge**
- **Vertices**

Detailed content of the Lecture:

- Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees.
- A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree.
- The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

Below are the steps for finding MST using Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is n ot formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

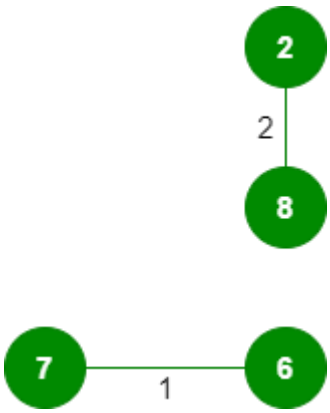


The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.
After sorting:

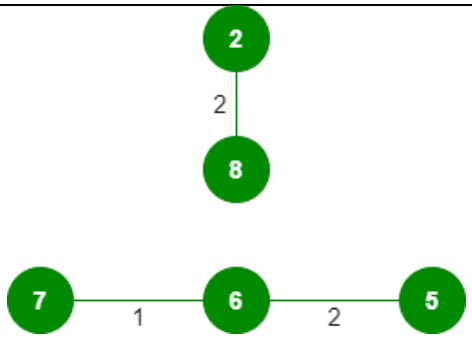
Weight	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from the sorted list of edges
1. Pick edge 7-6: No cycle is formed, include it.

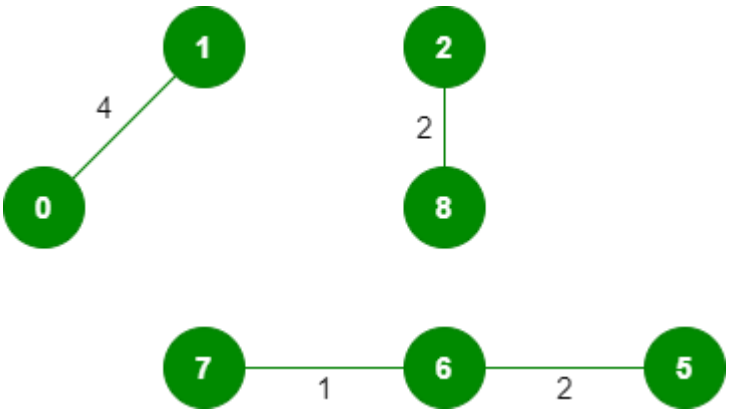
2. Pick edge 8-2: No cycle is formed, include it.



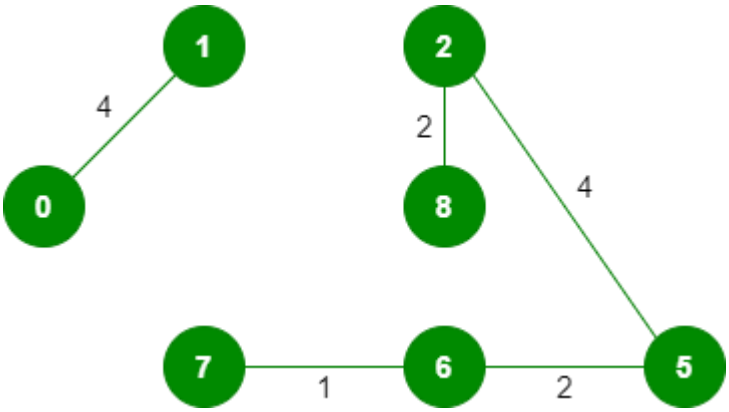
3. Pick edge 6-5: No cycle is formed, include it.



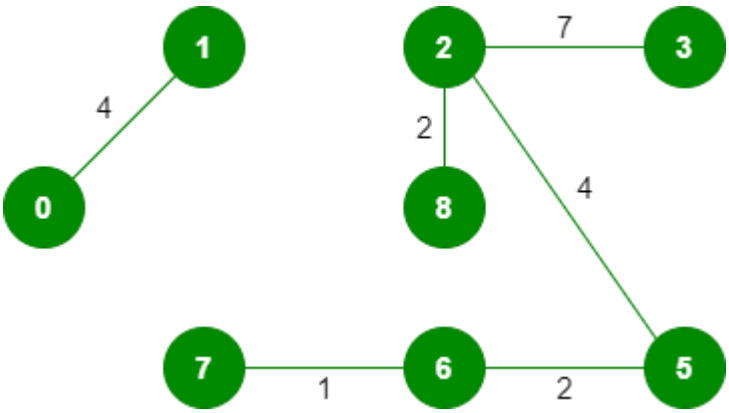
4. Pick edge 0-1: No cycle is formed, include it.



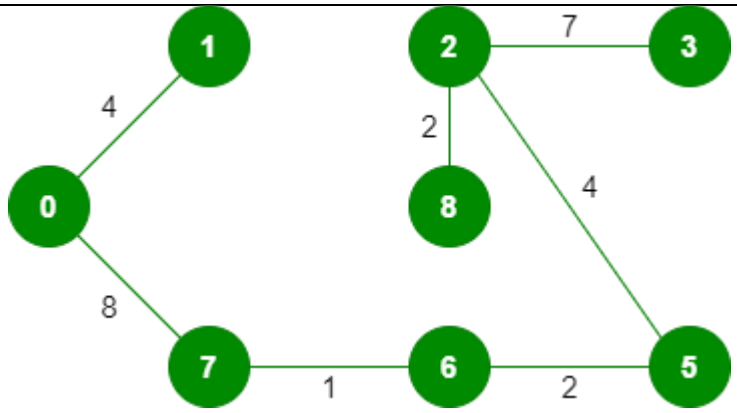
5. Pick edge 2-5: No cycle is formed, include it.



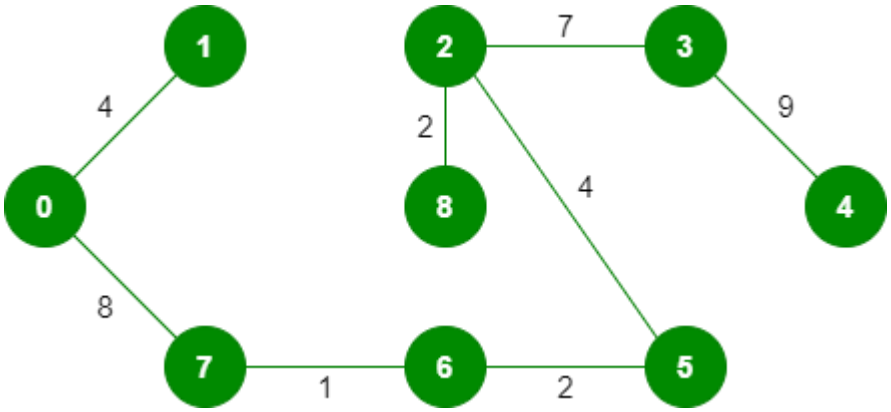
6. Pick edge 8-6: Since including this edge results in the cycle, discard it.
7. Pick edge 2-3: No cycle is formed, include it.



8. Pick edge 7-8: Since including this edge results in the cycle, discard it.
9. Pick edge 0-7: No cycle is formed, include it.



10. Pick edge 1-2: Since including this edge results in the cycle, discard it.
11. Pick edge 3-4: No cycle is formed, include it.



Since the number of edges included equals (V – 1), the algorithm stops here.

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

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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

L - 34

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

Topic of Lecture: Shortest path algorithms -Dijkstra's algorithm

Introduction :

- **Dijkstra's algorithm** is the iterative algorithmic process to provide us with the shortest path from one specific starting node to all other nodes of a graph. It is different from the minimum spanning tree as the shortest distance among two vertices might not involve all the vertices of the graph.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Tree
- Graph
- Vertex
- Vertices

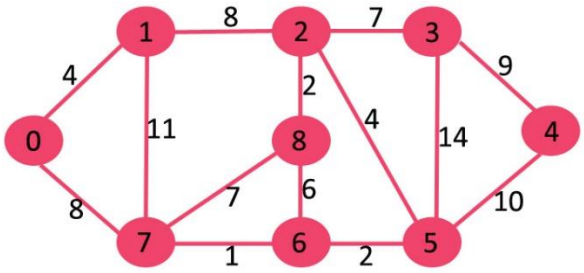
Detailed content of the Lecture:

Given a graph and a source vertex in the graph, find the shortest paths from the source to all vertices in the given graph.
Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a *SPT (shortest path tree)* with a given source as a root. We maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, we find a vertex that is in the other set (set of not yet included) and has a minimum distance from the source.
Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

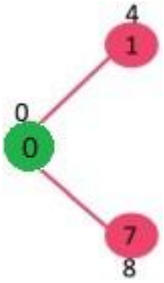
Algorithm

1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in the shortest-path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3) While *sptSet* doesn't include all vertices
….**a)** Pick a vertex u which is not there in *sptSet* and has a minimum distance value.
….**b)** Include u to *sptSet*.
….**c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

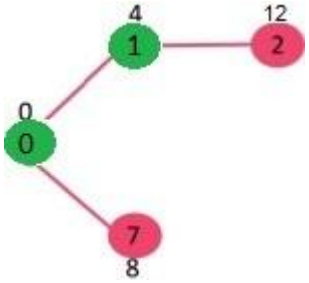
Let us understand with the following example:



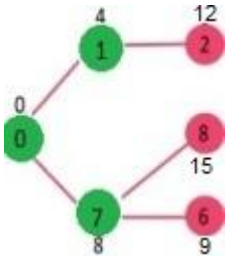
The set *sptSet* is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with a minimum distance value. The vertex 0 is picked, include it in *sptSet*. So *sptSet* becomes {0}. After including 0 to *sptSet*, update distance values of its adjacent vertices. Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. The following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green colour.



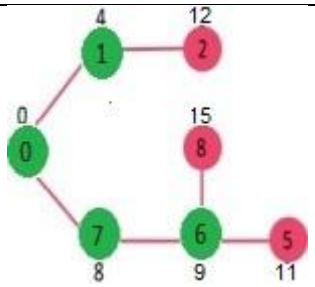
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



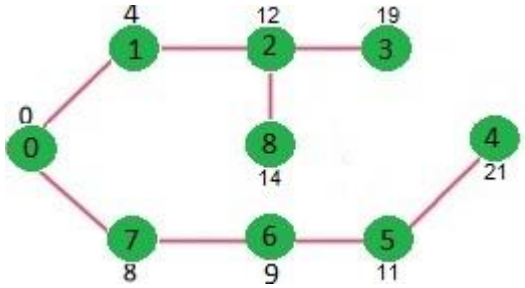
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



We repeat the above steps until *sptSet* includes all vertices of the given graph. Finally, we get the following Shortest Path Tree (SPT).



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

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
https://www.youtube.com/watch?v=pSqmAO-m7Lk

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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

L - 35

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

Topic of Lecture: Floyd Warshall algorithm

Introduction :

- The Floyd Warshall Algorithm is for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

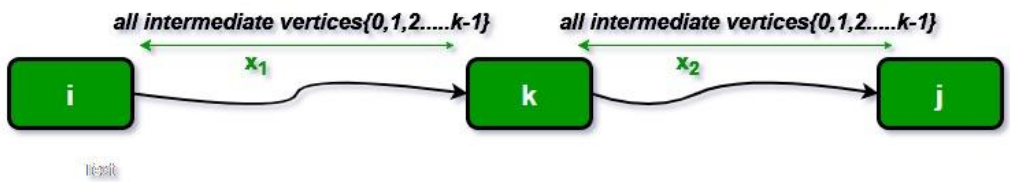
Prerequisite knowledge for Complete understanding and learning of Topic:

- **Tree**
- **Graph**

Detailed content of the Lecture:

The Floyd Warshall Algorithm is for solving the All Pairs Shortest Path problem.
The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

We initialize the solution matrix same as the input graph matrix as a first step.
Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices. For every pair (i, j) of the source and destination vertices respectively, there are two possible cases.
1) k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
2) k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]
The following figure shows the above optimal substructure property in the all-pairs shortest path problem.



```
// C++ Program for Floyd Warshall Algorithm
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough
value.This value will be used for
vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall(int graph[][V])
{
	/* dist[][] will be the output matrix
	that will finally have the shortest
	distances between every pair of vertices */
	int dist[V][V], i, j, k;

	/* Initialize the solution matrix same
	as input graph matrix. Or we can say
	the initial values of shortest distances
	are based on shortest paths considering
	no intermediate vertex. */
	for (i = 0; i < V; i++)
		for (j = 0; j < V; j++)
			dist[i][j] = graph[i][j];

	/* Add all vertices one by one to
	the set of intermediate vertices.
	---> Before start of an iteration,
	we have shortest distances between all
	pairs of vertices such that the
	shortest distances consider only the
	vertices in set {0, 1, 2, .. k-1} as
	intermediate vertices.
	----> After the end of an iteration,
	vertex no. k is added to the set of
	intermediate vertices and the set becomes {0, 1, 2, ..
	k} */
	for (k = 0; k < V; k++) {
		// Pick all vertices as source one by one
		for (i = 0; i < V; i++) {
			// Pick all vertices as destination for the
			// above picked source
			for (j = 0; j < V; j++) {
				// If vertex k is on the shortest path from
				// i to j, then update the value of
				// dist[i][j]
				if (dist[i][j] > (dist[i][k] + dist[k][j])
					&& (dist[k][j] != INF
						&& dist[i][k] != INF))
```

```
                                dist[i][j] = dist[i][k] + dist[k][j];
                        }
                }
        }

        // Print the shortest distance matrix
        printSolution(dist);
}

/* A utility function to print solution */
void printSolution(int dist[][V])
{
        cout << "The following matrix shows the shortest "
                        "distances"
                        " between every pair of vertices \n";
        for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                        if (dist[i][j] == INF)
                                cout << "INF"
                                        << "     ";
                        else
                                cout << dist[i][j] << "  ";
                }
                cout << endl;
        }
}

// Driver code
int main()
{
        /* Let us create the following weighted graph
                        10
        (0)------->(3)
                |          /|\
        5 |       |
                |          | 1
        \|/        |
        (1)------->(2)
                        3          */
        int graph[V][V] = { { 0, 5, INF, 10 },
                                                { INF, 0, 3, INF },
                                                { INF, INF, 0, 1 },
                                                { INF, INF, INF, 0 } };

        // Print the solution
        floydWarshall(graph);
        return 0;
}
```

Output :

```
Following matrix shows the shortest distances between every pair of vertices
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
```

```
    INF     INF     INF       0
```

Time Complexity: O(V^3)

The above program only prints the shortest distances. We can modify the solution to print the shortest paths also by storing the predecessor information in a separate 2D matrix. Also, the value of INF can be taken as INT_MAX from limits.h to make sure that we handle maximum possible value. When we take INF as INT_MAX, we need to change the if condition in the above program to avoid arithmetic overflow.

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

https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/
https://www.youtube.com/watch?v=NdBHw5mqIZE

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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

L - 36

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : IV – Graphs **Date of Lecture:**

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.

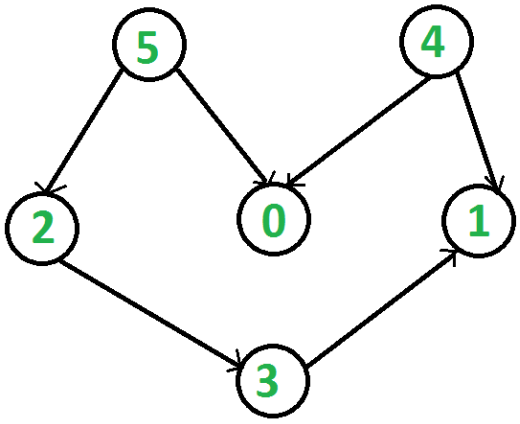
Prerequisite knowledge for Complete understanding and learning of Topic:

- **Tree**
- **Graph**

Detailed content of the Lecture:

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).



Topological Sorting vs Depth First Traversal (DFS):

In DFS, we print a vertex and then recursively call DFS for its adjacent vertices. In topological sorting, we need to print a vertex before its adjacent vertices. For example, in the given graph, the vertex ‘5’

should be printed before vertex ‘0’, but unlike DFS, the vertex ‘4’ should also be printed before vertex ‘0’. So Topological sorting is different from DFS. For example, a DFS of the shown graph is “5 2 3 1 0 4”, but it is not a topological sorting.

```
// A C++ program to print topological
// sorting of a DAG
#include <iostream>
#include <list>
#include <stack>
using namespace std;

// Class to represent a graph
class Graph {
	// No. of vertices'
	int V;

	// Pointer to an array containing adjacency listsList
	list<int>* adj;

	// A function used by topologicalSort
	void topologicalSortUtil(int v, bool visited[],
                                                stack<int>& Stack);

public:
	// Constructor
	Graph(int V);

	// function to add an edge to graph
	void addEdge(int v, int w);

	// prints a Topological Sort of
	// the complete graph
	void topologicalSort();
};

Graph::Graph(int V)
{
	this->V = V;
	adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
	// Add w to v’s list.
	adj[v].push_back(w);
}

// A recursive function used by topologicalSort
void Graph::topologicalSortUtil(int v, bool visited[],
                                                stack<int>& Stack)
{
	// Mark the current node as visited.
	visited[v] = true;

	// Recur for all the vertices
	// adjacent to this vertex
	list<int>::iterator i;
```

```
        for (i = adj[v].begin(); i != adj[v].end(); ++i)
                if (!visited[*i])
                        topologicalSortUtil(*i, visited, Stack);

        // Push current vertex to stack
        // which stores result
        Stack.push(v);
}

// The function to do Topological Sort.
// It uses recursive topologicalSortUtil()
void Graph::topologicalSort()
{
        stack<int> Stack;

        // Mark all the vertices as not visited
        bool* visited = new bool[V];
        for (int i = 0; i < V; i++)
                visited[i] = false;

        // Call the recursive helper function
        // to store Topological
        // Sort starting from all
        // vertices one by one
        for (int i = 0; i < V; i++)
                if (visited[i] == false)
                        topologicalSortUtil(i, visited, Stack);

        // Print contents of stack
        while (Stack.empty() == false) {
                cout << Stack.top() << " ";
                Stack.pop();
        }
}

// Driver Code
int main()
{
        // Create a graph given in the above diagram
        Graph g(6);
        g.addEdge(5, 2);
        g.addEdge(5, 0);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(2, 3);
        g.addEdge(3, 1);

        cout << "Following is a Topological Sort of the given "
                        "graph \n";

        // Function Call
        g.topologicalSort();

        return 0;
}
```

Output:

```
Following is a Topological Sort of the given graph
5 4 2 3 1 0
```

Complexity Analysis:

- **Time Complexity:** O(V+E).
  The above algorithm is simply DFS with an extra stack. So time complexity is the same as DFS which is.
- **Auxiliary space:** O(V).
  The extra space is needed for the stack.

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

https://www.geeksforgeeks.org/topological-sorting/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 37

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: Searching - Linear Search

Introduction :

- Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Linear Search
- Binary Search

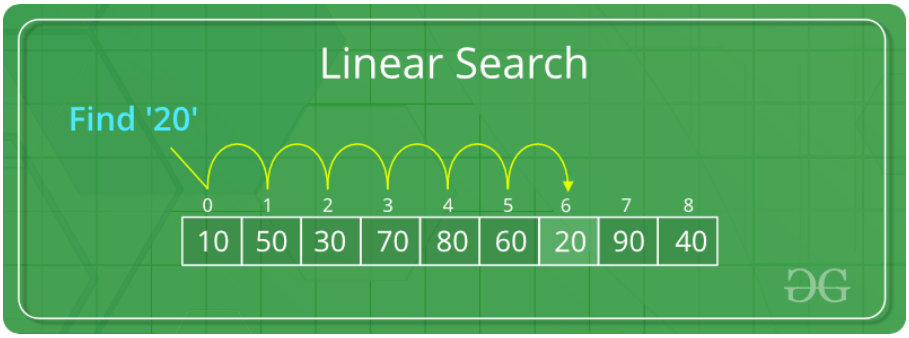
Detailed content of the Lecture:

Based on the type of search operation, these algorithms are generally classified into two categories:

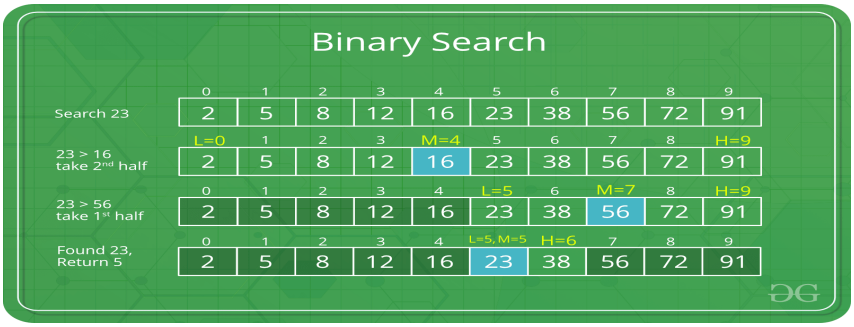
1. **Sequential Search**: In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search.
2. **Interval Search**: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: Binary Search.

A simple approach is to do a **linear search**, i.e

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
- If x matches with an element, return the index.
- If x doesn't match with any of elements, return -1.



Binary Search to find the element “23” in a given list of numbers



Problem: Given an array arr[] of n elements, write a function to search a given element x in arr[].

Examples :

Input : arr[] = {10, 20, 80, 30, 60, 50,
110, 100, 130, 170}
x = 110;
Output : 6

Element x is present at index 6

Input : arr[] = {10, 20, 80, 30, 60, 50,
110, 100, 130, 170}
x = 175;
Output : -1
Element x is not present in arr[].

```
// C++ code to linearly search x in arr[]. If x
// is present then return its location, otherwise
// return -1

#include <iostream>
using namespace std;

int search(int arr[], int n, int x)
{
        int i;
        for (i = 0; i < n; i++)
                if (arr[i] == x)
                        return i;
        return -1;
}

// Driver code
int main(void)
{
        int arr[] = { 2, 3, 4, 10, 40 };
        int x = 10;
        int n = sizeof(arr) / sizeof(arr[0]);

        // Function call
        int result = search(arr, n, x);
        (result == -1)
                ? cout << "Element is not present in array"
                : cout << "Element is present at index " << result;
```

```
    return 0;
}
```

Output:

```
Element is present at index 3
```

The time complexity of the above algorithm is O(n).

- Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster-searching comparison to Linear search.

Improve Linear Search Worst-Case Complexity

1. if element Found at last O(n) to O(1)
2. It is the same as previous method because here we are performing 2 'if' operations in one iteration of the loop and in previous method we performed only 1 'if' operation. This makes both the time complexities same.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/linear-search/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: Binary Search

Introduction :

- Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one. We used binary search in the guessing game in the introductory tutorial.

Prerequisite knowledge for Complete understanding and learning of Topic:

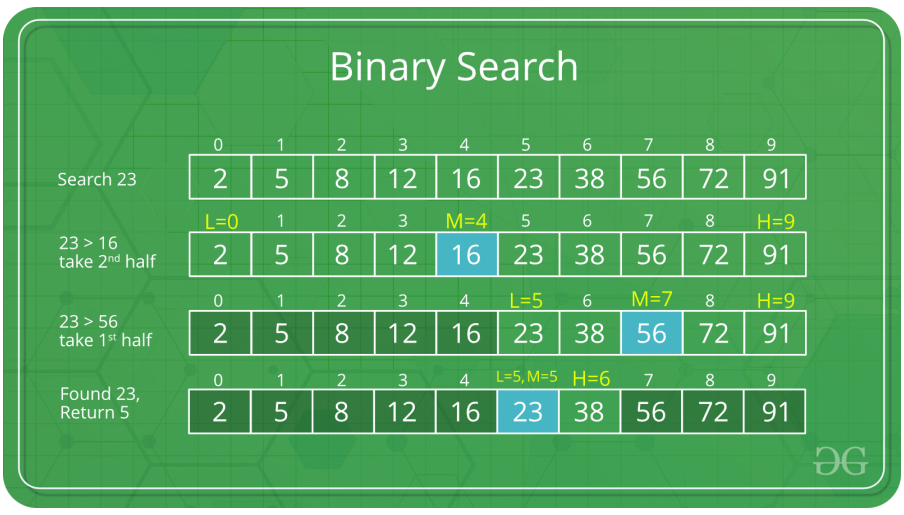
- Searching
- Binary Search

Detailed content of the Lecture:

- A simple approach is to do a **linear search**. The time complexity of the above algorithm is O(n). Another approach to perform the same task is using Binary Search.

Binary Search:

- Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.



The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

// C++ program to implement recursive Binary Search

```
#include <bits/stdc++.h>
using namespace std;

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
	if (r >= l) {
		int mid = l + (r - l) / 2;

		// If the element is present at the middle
		// itself
		if (arr[mid] == x)
			return mid;

		// If element is smaller than mid, then
		// it can only be present in left subarray
		if (arr[mid] > x)
			return binarySearch(arr, l, mid - 1, x);

		// Else the element can only be present
		// in right subarray
		return binarySearch(arr, mid + 1, r, x);
	}

	// We reach here when element is not
	// present in array
	return -1;
}

int main(void)
{
	int arr[] = { 2, 3, 4, 10, 40 };
	int x = 10;
	int n = sizeof(arr) / sizeof(arr[0]);
	int result = binarySearch(arr, 0, n - 1, x);
	(result == -1) ? cout << "Element is not present in array"
				: cout << "Element is present at index " << result;
	return 0;
}
```

Output :

```
Element is present at index 3
```

Time Complexity:
The time complexity of Binary Search can be written as

$T(n) = T(n/2) + c$

The above recurrence can be solved either using the Recurrence Tree method or Master method. It

falls in case II of the Master Method and the solution of the recurrence is .
Auxiliary Space: O(1) in case of iterative implementation. In the case of recursive implementation, O(Logn) recursion call stack space.

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

https://www.geeksforgeeks.org/binary-search/
https://www.youtube.com/watch?v=P3YID7liBug

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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

L - 39

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: Sorting - Bubble sort

Introduction :

- A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Sorting
- List

Detailed content of the Lecture:

- A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

For example:

- The below list of characters is sorted in increasing order of their ASCII values. That is, the character with lesser ASCII value will be placed first than the character with higher ASCII value.

Bubble Sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:

First Pass:

(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4

(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2

(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does

not swap them.

Second Pass:
(**1 4** 2 5 8) –> (**1 4** 2 5 8)
(1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2
(1 2 **4 5** 8) –> (1 2 **4 5** 8)
(1 2 4 **5 8**) –> (1 2 4 **5 8**)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:
(**1 2** 4 5 8) –> (**1 2** 4 5 8)
(1 **2 4** 5 8) –> (1 **2 4** 5 8)
(1 2 **4 5** 8) –> (1 2 **4 5** 8)
(1 2 4 **5 8**) –> (1 2 4 **5 8**)

Worst and Average Case Time Complexity:

O(n*n). Worst case occurs when array is reverse sorted.

Best Case Time Complexity:

O(n). Best case occurs when array is already sorted.

AuxiliarySpace: O(1)

Boundary Cases:

Bubble sort takes minimum time (Order of n) when elements are already sorted.

SortingInPlace: Yes

Stable: Yes

Due to its simplicity, bubble sort is often used to introduce the concept of a sorting algorithm.

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

https://www.geeksforgeeks.org/bubble-sort/

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 40

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: Selection sort - Insertion sort - Merge sort

Introduction :

- 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.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Sorting
- List

Detailed content of the Lecture:

Selection sort

- 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.

Time Complexity of InsertionSort

- **Best Case : $O(n^2)$** #Means array is already sorted.
- **Average Case : $O(n^2)$** #Means array with random numbers.
- **Worst Case : $O(n^2)$** #Means array with descending order.

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

Insertion Sort:

- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.
- **Algorithm**

  To sort an array of size n in ascending order:

  1: Iterate from arr[1] to arr[n] over the array.

  2: Compare the current element (key) to its predecessor.

  3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Time Complexity of InsertionSort

- **Best Case : O(n)** #Means array is already sorted.
- **Average Case : O(n^2)** #Means array with random numbers.
- **Worst Case : O(n^2)** #Means array with descending order.

Example:

Let us sort the following numbers using Insertion sort mechanism,

12, 11, 13, 5, 15

Let us loop for i = 1 (second element of the array) to 4 (last element of the array)

i = 1. Since 11 is smaller than 12, move 12 and insert 11 before 12

11, 12, 13, 5, 15

i = 2. 13 will remain at its position as all elements in A[0..I-1] are smaller than 13

11, 12, 13, 5, 15

i = 3. 5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position.

5, 11, 12, 13, 15

i = 4. 15 will be at position 5 as all the elements are smaller than 15.

5, 11, 12, 13,15

Merge Sort:

- One of the best sorting technique. If n value is large, it follows divide and conquer approach.
- Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves.

Time Complexity :

- Best Case : **O(nlogn)** #Means array is already sorted.
- Average Case : **O(nlogn)** #Means array with random numbers.
- Worst Case : **O(nlogn)** #Means array with descending order.

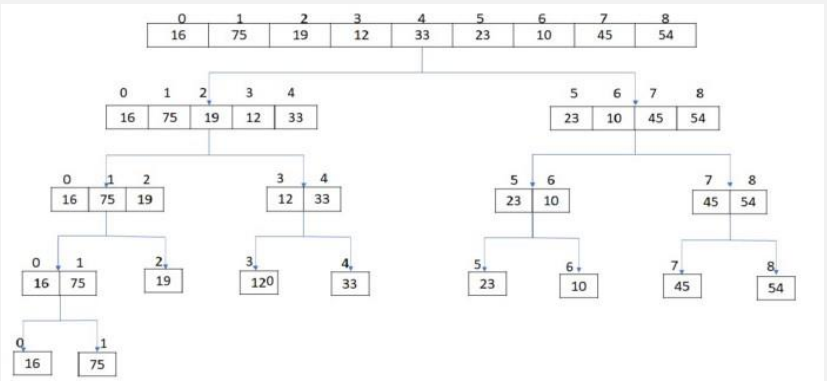
Algorithm:

1. Find the middle point to divide the array into two halves:
middle m = (l+r)/2
2. Call mergeSort for first half:
Call mergeSort(arr, l, m)
3. Call mergeSort for second half:
Call mergeSort(arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
Call merge(arr, l, m, r)

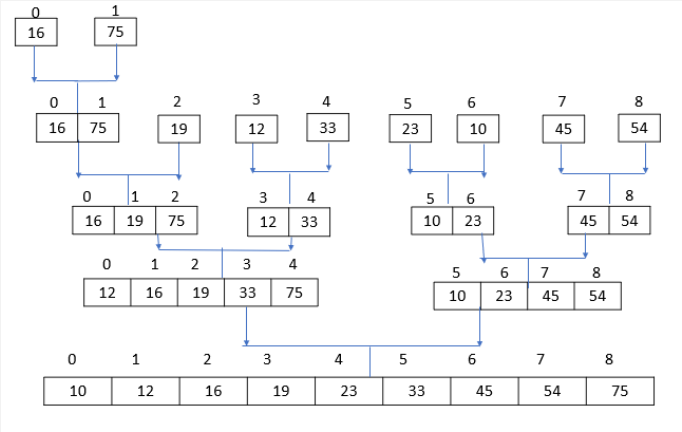
Example:

Let us take an array of elements 16,75,19,12,33,2,310,45,54

Partition Mechanism :



Merging Mechanism:



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

https://www.geeksforgeeks.org/insertion-sort/
https://www.geeksforgeeks.org/merge-sort/
https://www.geeksforgeeks.org/selection-sort/

Important Books/Journals for further learning including the page nos.:
Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 41

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: Hashing - Hash Functions

Introduction :

- A hash algorithm is **a function that converts a data string into a numeric string output of fixed length**. The output string is generally much smaller than the original data. ... Two of the most common hash algorithms are the MD5 (Message-Digest algorithm 5) and the SHA-1 (Secure Hash Algorithm).

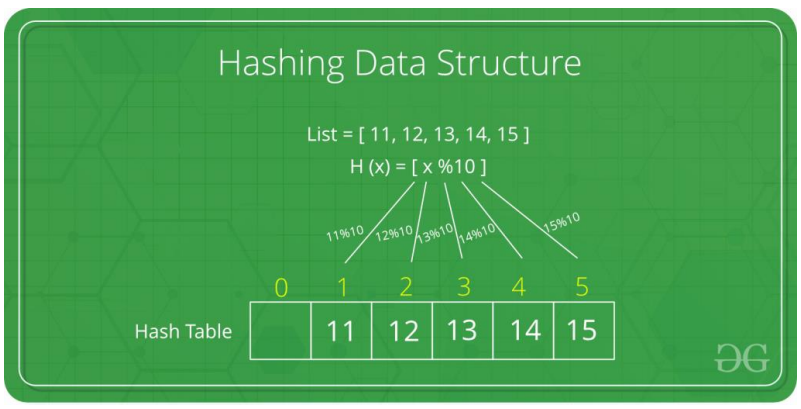
Prerequisite knowledge for Complete understanding and learning of Topic:

- Conversion of String
- Data Conversion

Detailed content of the Lecture:

- Hashing is a technique or process of mapping keys, values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

- Let a hash function H(x) maps the value at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



Suppose we want to design a system for storing employee records keyed using phone numbers. And we want following queries to be performed efficiently:

1. Insert a phone number and corresponding information.

2. Search a phone number and fetch the information.
3. Delete a phone number and related information.

We can think of using the following data structures to maintain information about different phone numbers.

1. Array of phone numbers and records.
2. Linked List of phone numbers and records.
3. Balanced binary search tree with phone numbers as keys.
4. Direct Access Table.
   - For **arrays and linked lists**, we need to search in a linear fashion, which can be costly in practice. If we use arrays and keep the data sorted, then a phone number can be searched in O(Logn) time using Binary Search, but insert and delete operations become costly as we have to maintain sorted order.

   - With **balanced binary search tree**, we get moderate search, insert and delete times. All of these operations can be guaranteed to be in O(Logn) time.
   - Another solution that one can think of is to use a **direct access table** where we make a big array and use phone numbers as index in the array.
   - An entry in array is NIL if phone number is not present, else the array entry stores pointer to records corresponding to phone number.
   - Time complexity wise this solution is the best among all, we can do all operations in O(1) time.
   - For example to insert a phone number, we create a record with details of given phone number, use phone number as index and store the pointer to the created record in table. This solution has many practical limitations.
   - First problem with this solution is extra space required is huge.
   - For example if phone number is n digits, we need $O(m * 10^n)$ space for table where m is size of a pointer to record.
   - Another problem is an integer in a programming language may not store n digits.

Hash Table:
   - An array that stores pointers to records corresponding to a given phone number. An entry in hash table is NIL if no existing phone number has hash function value equal to the index for the entry.

Collision Handling:
   - Since a hash function gets us a small number for a big key, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique. Following are the ways to handle collisions:

- **Chaining:**
  - The idea is to make each cell of hash table point to a linked list of records that have same hash function value. Chaining is simple, but requires additional memory outside the table.
- **Open Addressing:**
  - In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we examine the table slots one by one until the desired element is found or it is clear that the element is not in the table.

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

https://www.geeksforgeeks.org/hashing-data-structure/

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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

L - 42

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: Separate Chaining - Open Addressing

Introduction :

- Cache performance of chaining is not good as keys are stored using linked list.
- Open addressing provides better cache performance as everything is stored in the same table.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Linked List
- Sorting

Detailed content of the Lecture:

Open Addressing

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

Insert(k):

- Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search(k):

- Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete(k):

- **Delete operation is interesting**. If we simply delete a key, then the search may fail. So slots of deleted keys are marked specially as "deleted". The insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Open Addressing is done in the following ways:

a) ***Linear Probing:***

- In linear probing, we linearly probe for next slot. For example, the typical gap between two probes is 1 as seen in the example below.
  Let **hash(x)** be the slot index computed using a hash function and **S** be the table size

If slot hash(x) % S is full, then we try (hash(x) + 1) % S

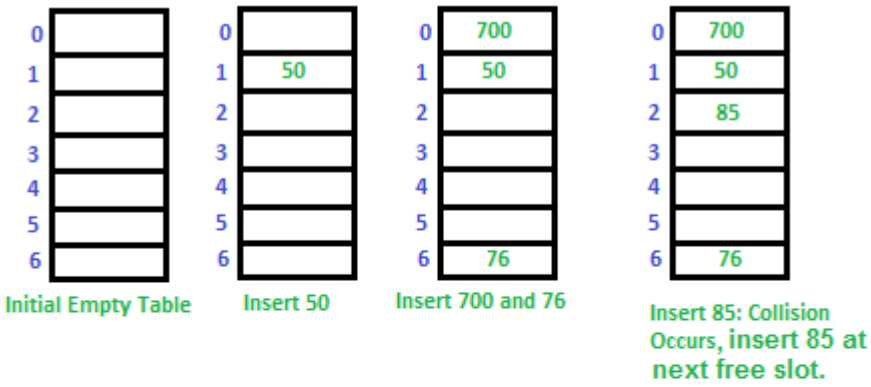
If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S

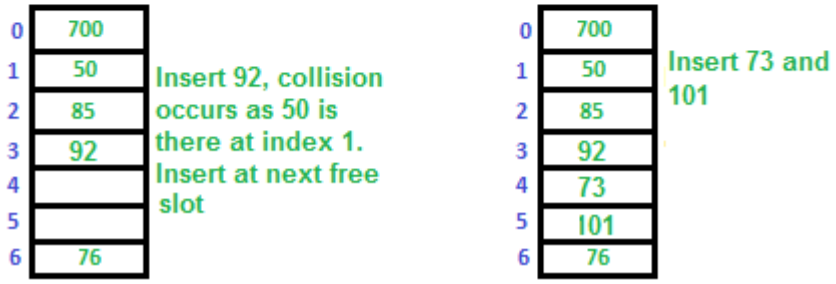
If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S

.....

.....

Let us consider a simple hash function as “key mod 7” and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.





Challenges in Linear Probing :

1. **Primary Clustering:**
   - One of the problems with linear probing is Primary clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search for an element.
2. **Secondary Clustering:**
   - Secondary clustering is less severe, two records only have the same collision chain (Probe Sequence) if their initial position is the same.

b) Quadratic Probing

- We look for i^2‘th slot in i’th iteration.

let hash(x) be the slot index computed using hash function.

If slot hash(x) % S is full, then we try (hash(x) + 1*1) % S

If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S

If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S

c) Double Hashing

- We use another hash function hash2(x) and look for i*hash2(x) slot in i’th rotation.

let hash(x) be the slot index computed using hash function.

If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S

If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) % S

If (hash(x) + 2*hash2(x)) % S is also full, then we try (hash(x) + 3*hash2(x)) % S

S.No.	**Separate Chaining**	**Open Addressing**
1.	Chaining is Simpler to implement.	Open Addressing requires more computation.
2.	In chaining, Hash table never fills up, we can always add more elements to chain.	In open addressing, table may become full.
3.	Chaining is Less sensitive to the hash function or load factors.	Open addressing requires extra care to avoid clustering and load factor.
4.	Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.	Open addressing is used when the frequency and number of keys is known.
5.	Cache performance of chaining is not good as keys are stored using linked list.	Open addressing provides better cache performance as everything is stored in the same table.
6.	Wastage of Space (Some Parts of hash table in chaining are never used).	In Open addressing, a slot can be used even if an input doesn't map to it.
7.	Chaining uses extra space for links.	No links in Open addressing

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

https://www.geeksforgeeks.org/hashing-set-3-open-addressing/

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 43

AI & DS	**LECTURE HANDOUTS**	**II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: Rehashing - Extendible Hashing

Introduction :

- **Extendible Hashing** is a dynamic hashing method wherein directories, and buckets are used to hash data. It is an aggressively flexible method in which the hash function also experiences dynamic changes.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Hash Function
- Buckets
- Directories

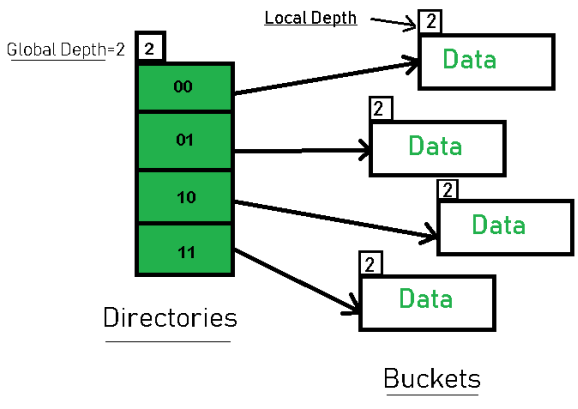
Detailed content of the Lecture:

Main features of Extendible Hashing:

The main features in this hashing technique are:

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

Basic Structure of Extendible Hashing:



Extendible Hashing

Frequently used terms in Extendible Hashing:

- **Directories:** These containers store pointers to buckets. Each directory is given a unique id which may change each time when expansion takes place. The hash function returns this directory id which is used to navigate to the appropriate bucket. Number of Directories = 2^Global Depth.
- **Buckets:** They store the hashed keys. Directories point to buckets. A bucket may contain more than one pointers to it if its local depth is less than the global depth.
- **Global Depth:** It is associated with the Directories. They denote the number of bits which are used by the hash function to categorize the keys. Global Depth = Number of bits in directory id.
- **Local Depth:** It is the same as that of Global Depth except for the fact that Local Depth is associated with the buckets and not the directories. Local depth in accordance with the global depth is used to decide the action that to be performed in case an overflow occurs. Local Depth is always less than or equal to the Global Depth.
- **Bucket Splitting:** When the number of elements in a bucket exceeds a particular size, then the bucket is split into two parts.
- **Directory Expansion:** Directory Expansion Takes place when a bucket overflows. Directory Expansion is performed when the local depth of the overflowing bucket is equal to the global depth.

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

https://www.geeksforgeeks.org/extendible-hashing-dynamic

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

Mark Allen Weiss, Data structure and Algorithm Analysis in C++, Pearson India,2012, **page nos:** 379

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 - 44

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: File Storage Concepts : Sequential Access Method (SAM)

Introduction :

- Sequential access memory (SAM) is **a class of data storage devices that read stored data in a sequence**. This is in contrast to random access memory (RAM) where data can be accessed in any order. Sequential access devices are usually a form of magnetic storage or optical storage.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Files
- RAM
- Storage

Detailed content of the Lecture:

File Storage Concepts

A file is a sequence of records. File organization refers to physical layout or a structureof record occurrences in a file. File organization determines the way records arestored and accessed.

In many cases, all records in a file are of the same record type. If every record in the file has exactly the same size (in bytes), the file is said to be made of fixed-lengthrecords. If different records in the file have different sizes, the file is said to be made up of variable-length records. A file may have variable-length records for severalreasons :

i) The file records are of the same record type, but one or more fields are ofvarying sizes (variable-length fields).

ii) The file records are of the same record type but one or more fields may havemultiple values for individual records. Such a field is called repeating field and agroup of values for the field is often called a repeating group.

iii) The file records are of the same record type, but one or more fields are optional.

iv) The file has records of different record types and hence of varying size (mixedfile). This would occur if related records of different types were clustered (placedtogether) on disk blocks.

Sequential Access Method (SAM)

- In sequential files, the records are stored in a predefined order.
- Records whichoccur in a sequential file are usually sorted on the primary key and physically arranged on the storage medium in order by primary key.

- If only sequential access is required (which is rarely the case), sequential media (magnetic tapes) are suitable and probablythe most cost-effective way of processing such files.
- Direct access devices such asdisks may be but are not necessarily, referenced sequentially. Some types of processing are best done through sequential access, even when direct access devicesare used.

➢ Sequential access is fast and efficient while dealing with large volumes of data thatneed to be processed periodically. However, it is require that all new transactionsbe sorted into a proper sequence for sequential access processing. Also, most ofthe database or file may have to be searched to locate, store, or modify even a smallnumber of data records. Thus, this method is too slow to handle applications requiring immediate updating or responses.

➢ Sequential files are generally used for backup or transporting data to a differentsystem. A sequential ASCII file is a popular export/import format that most databasesystems support.

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

http://egyankosh.ac.in/bitstream/123456789/25571/1/Unit-8.pdf

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

http://www.tmv.edu.in/pdf/Distance_education/BCA%20Books/BCA%20III%20SEM/BCA-322%20DBMS.pdf

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

AI & DS | **LECTURE HANDOUTS** | **II/III**

Course Name with Code : 19ADC01-Data Structures and Files

Course Faculty : M.S.Soundarya

Unit : V-Searching,Sorting,Hashing And Files **Date of Lecture:**

Topic of Lecture: Indexed Sequential Access Method (ISAM) – Direct Access Method (DAM)

Introduction :

- A file is a sequence of records. File organization refers to physical layout or a structure of record occurrences in a file. File organization determines the way records arestored and accessed.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Files
- Organization

Detailed content of the Lecture:

Indexed Sequential Access Method (ISAM)

- In indexed sequential files, record occurrences are sorted and stored in order byprimary key on a direct access storage device.
- In addition, a separate table (or file) called an index is maintained on primary key values to give the physical address ofeach record occurrence. This approach gives (almost) direct access to record occurrences via the index table and sequential access via the way in which the records are laid out on the storage medium.

- The physical address of a record given by the index file is also called a pointer. Thepointer or address can take many forms depending on the operating system and the database one is using.
- Today systems use virtual addresses instead of physical addresses. A virtual address could be based on imaginary disk drive layout.
- The database refers to a base set of tracks and cylinders. The computer then maps these values into actual storagelocations. This arrangement is the basis for an approach known as the virtualsequential access method (VSAM).
- Another common approach is to define alocation in terms of its distance from the start of a file (relative address). Virtual orrelative addresses are always better than the physical address because of their portability.

In case a few records need to be processed quickly, the index is used to directlyaccess the records needed. However, when large numbers of records must be processed periodically, the sequential organization provided by this method is used.

An illustration of access using index file is given in Fig. 2.8.

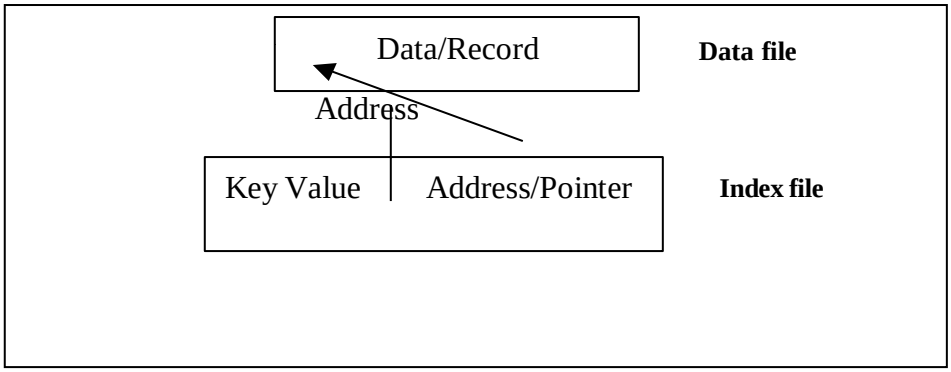


Fig. 2.8: Data access using index file

Direct Access Method (DAM)

- When using the direct access method, the record occurrences in a file do not haveto be arranged in any particular sequence on storage media.
- However, the computermust keep track of the storage location of each record using a variety of directorganization methods so that data is retrieved when needed.
- New transactions datado not have to be sorted, and processing that requires immediate responses orupdating is easily handled.

In the direct access method, an alogrithm is used to compute the address of a record.The primary key value is the input to the algorithm and the block address of therecord is the output.

To implement the approach, a portion of the storage space is reserved for the file.This space should be large enough to hold the file plus some allowance for growth. Then the algorithm that generates the appropriate address for a given primary key isdevised. The algorithm is commonly called hashing algorithm. The process ofconverting primary key values into addresses is called key-to-address transformation.

More than one logical record usually fits into a block, so we may think of the reserved storage area as being broken into record slots sequentially numbered from 1 to n.These sequential numbers are called relative pointers or relative addresses, becausethey indicate the position of the record relative to the beginning of the file.

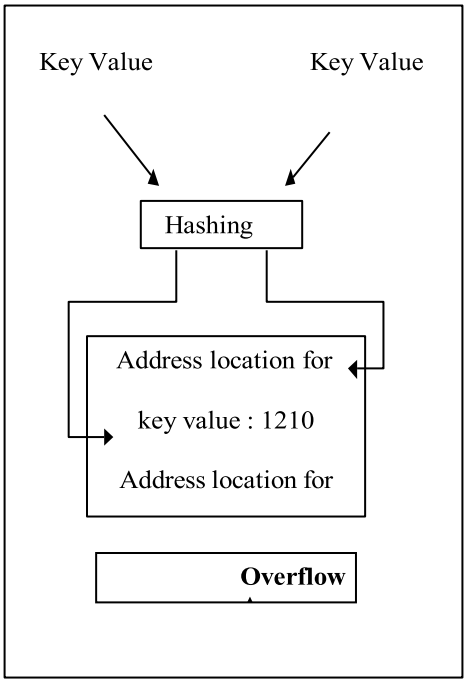
The objective of the hashing algorithm is to generate relative addresses that dispersethe records throughout the reserved storage space in a random but uniform manner.The records can be retrieved very rapidly because the address is computed ratherthan found through table look-up via indexes stored on a disk file.

A collision is said to occur if more than one record maps to the same block. Sinceone block usually holds several records, collisions are only a problem when thenumber of records mapping to a block exceeds the block's capacity. To account forthis event, most direct access methods support an overflow area for collisions which is searched sequentially.

The hashed key approach is extremely fast since the key's value is immediatelyconverted into a storage location, and data can be retrieved in one pass to the disk.

An illustration of direct access method using hashed key is given in

Direct access using hashed-key approach



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

http://egyankosh.ac.in/bitstream/123456789/25571/1/Unit-8.pdf

Important Books/Journals for further learning including the page nos.:
http://www.tmv.edu.in/pdf/Distance_education/BCA%20Books/BCA%20III%20SEM/BCA-322%20DBMS.pdf

Course Faculty

Verified by HoD