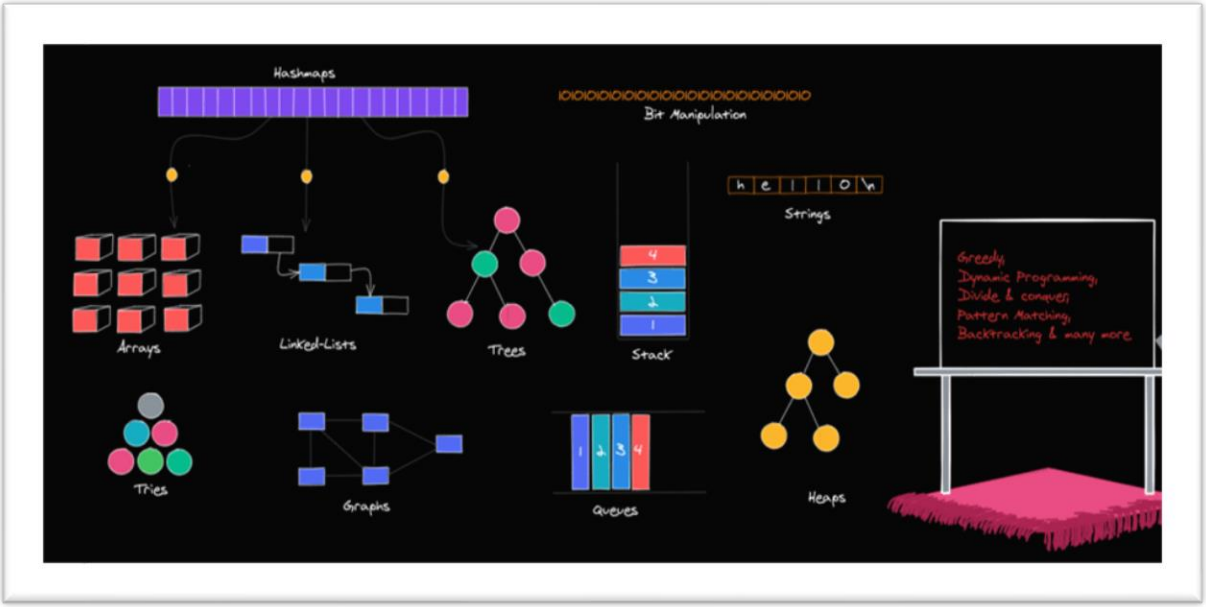
BHADRAK ENGINEERING SCHOOL & TECHNOLOGY
(BEST), ASURALI, BHADRAK

DATA STRUCTURE (TH-02)

(As per the 2020-21 syllabus of the SCTE&VT, Bhubaneswar, Odisha)



Third Semester

COMPUTER SCIENCE & ENGG.

Prepared By: Er. S.Palit

3rd SEMESTER COMPUTER SCIENCE & ENGG.
DATA STRUCTURE(TH_02)
Concerned Lecturer: Er. S.Palit
Mark Distribution

Sl. No.	Topic	Expected Marks
1.	**Introduction**	**05**
2.	**String Processing**	**05**
3.	**Arrays**	**15**
4.	**Stacks & Queues**	**20**
5.	**Linked List**	**15**
6.	**Tree**	**10**
7.	**Graphs**	**15**
8.	**Sorting Searching & Merging**	**15**
9.	**File Organization**	**10**
TOTAL		**110**

Chapter-01
Introduction

Articles to be covered

1.1. Concept of data, information & data type
1.2. Data Structure & explain different types of operation
1.3. Abstract Data type (ADT)
1.4. Algorithms & their Complexities
1.5. Space - time trade off

1.1. Concept of data, information & data type -:

Data:

- Data are simply Values or set of values .
- Data is represented in the form of text or number or in the form of fig. tables, graphics, pictures etc.

Information

. information is derived from data .
Date-> Method to process data –>information. **Some examples of data & information -:**

Information details Data
Name Muskan
Gender female

Ex-2-:

A B

Bus(1 hr)
Care (1/2hr) Bike (45 min) Cycle (5 hr) ,Rickshaw (8hr)

- The distance between the two cities is the data (160 km) ● The info helps a person to choose the mode of travel.
- One can decide to go by train or bus depending on the urgency. ● So the mode of travel given above are the information.

Data type-:A data takes how many bits of spaces in a memory location is known as data

type.

Int 2 bytes (16 bits)
Char 1 byte (8 bits)
float 4 bytes(32 bits)

1.2 Data Structure & explain different types of operation -

Data structure -

- It may be organized in different ways.
- The logical or mathematical organization of data is known as data structure.

Types of Data structure:

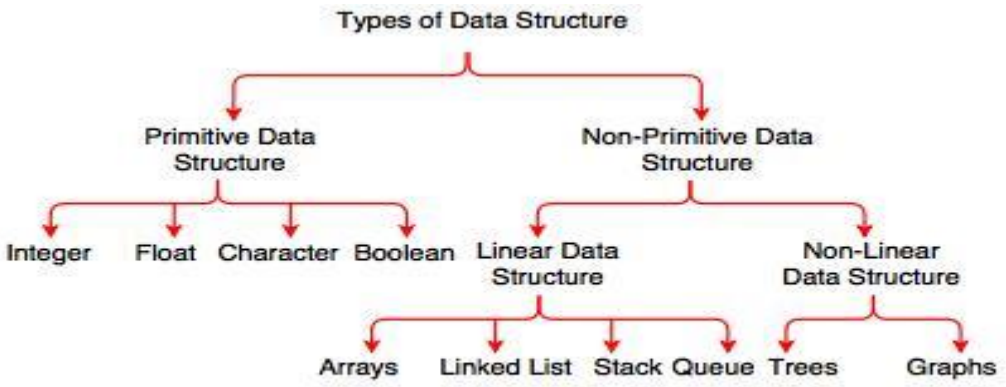


Fig. Types of Data Structure

Operations of data structure:

- Traverse- Accessing a record on element exactly once so that certain nodes in the record may be processed once.
- Searching- Find this location of an element in a record.
- Inserting -Adding a new item to data structure.
- Deleting- Remove an element from the data structure.
- Sorting- Arranging the items or elements in a record in some logical order .
- Merging :- Combining a Single or two different records into a single record.

1.3 Abstract Data type (ADT):-

- An abstract data type refers to a set of data values and associated operation that are specified accurately, independently and particular implementation.
- With an ADT, we know that a specific data type can do, but actually how it does work

that is hidden.

- ADT Consists of a self-definition that allows to use the function while hiding the implementation.
- Two basic Data structures mainly Array & Linked List can be used to implement ADT

1.4. Algorithms & their Complexities :

Algorithms :- An algorithm is a well-defined list for solving a particular problem.

Example :
Write an algorithm to check the greatest number among the three.

- Step-1 - Read the values of a, b, c
- Step-2 - Check whether a>b & a>c if yes then a is the greatest value among three .
- Step-3- Otherwise check b>a & b>c if yes then b is the greatest value among the three.
- Step-4- Otherwise c is the greatest value among them.

Example -Do by yourself.
Write an algorithm to check the smallest value among three numbers.

Complexities:

- Complexity of an algorithm is a function which gives the running time and space in terms of input Size.

Complexity type: - complexity are 2 types.

• Time Complexity -T(n) .

The time Complexity of an algorithms which gives running time in terms of input size.

• Space Complexity: - S(n)

- n= no. of element in DS.
- It is of an algorithm of a function which gives required space or running space in terms of input Size of the data Structure

1.5 : Space - time trade off:

Space time trade off refers to a choice between algorithmic solution of a data processing problem that allows one to decrease the running time of an algorithm by increasing the space to store the data vise-versa.

Possible Short Questions with Answers

Q1. Write the diff. bet data & Information?

Data:

- Data are simply Values or set of values.
- Data is represented in the form of text or number or in the form of fig. tables, graphics, pictures etc.

Information:

. Information is derived from data.

Date-> Method to process data –>information.

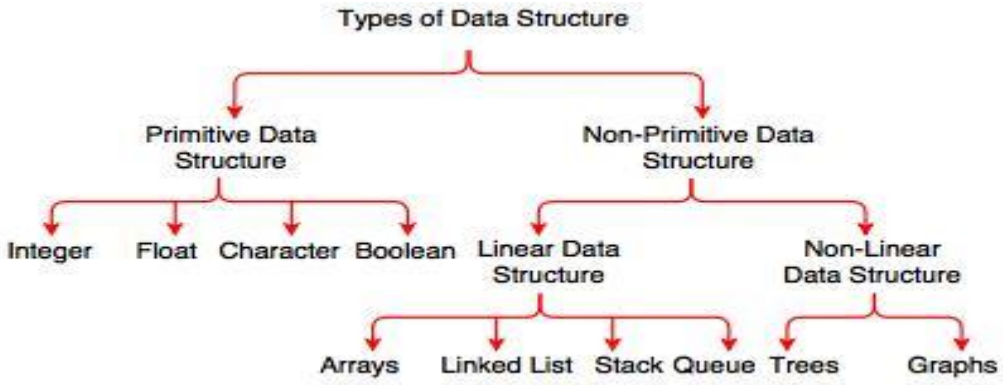
Q2. What do you mean by data Structure?

Data Structure:

- It may be organized in different ways.

The logical or mathematical organization of data is known as data structure.

Q3. Write down the types of DS?



Q4. Write down the different operation of DS?

Operations of data structure:

- Traverse- Accessing a record on element a exactly once so that I Certain dems in the record may be processed - 1

- Searching- Find this location of an element in a record. ● Inserting -

  Adding a new item to data structure.

- Deleting- Remove an element from the data structure.

- Sorting- Arranging the items or elements in a record in some logical order.
- Merging: - Combining a Single or two different records into a single record.

Q5. What is ADT?

- ADT stand for Abstract Data Type.
- With an ADT, we know that a specific data type can do, but actually how it does work that is hidden.
- ADT Consists of a self-definition that allows to use the function while hiding the implementation.

Long Questions:

1. Why is the algorithm used? Explain the Complexity of an algorithm and the space time trade off of an algorithm.

Chapter-02

String processing

Articles to be covered

2.1 Explain Basic Terminology, Storing Strings
2.2 State Character Data Type,
2.3 Discuss String Operations

2.1 basic terminology

String- Finite sequence of zero or more characters is called a string.

- It is denoted by S.
  - example- S= "best"
  - string with zero-character aur no character is known as null string.
    Ex- S=' '
  - The number of characters in a string is known as string length.
    Example- S=' BEST'
    string length. =4
    S=' '
    string length. =0

- Specific stream will be denoted by encroaching its characters in a single quotation mark.

Note

Space is considered as a character .

It is denoted by ___ .

s=__ =1

s=___=2

s='The end'=7

Concatenation of two strings-:

- The string consists of the characters of a string S1 followed by the characters of another string S2 is called concatenation of string S1 and S2.
  - Concatenation operation of two strings s1 & s2 is denoted by s1||s2 .
  - the length of S1||S2= the sum of length of string S1 and S2.
    Example- s1='BE' =2
    s2='ST'=2
    s1||s2='BEST'=4.

Substring :

- A string y is called a substring of s.If there exist string xz such that x||y ||z .
- If string X is called the initial substring of s, if Z is an Empty string then y is called a terminal sub- string of string ‘s’.
- Note- If Y is a substring of s then the length of Y never exceeds the length of s.

Storing string

Strings are stored in three types of structure.

1. Fixed length structure
2. variable length structure
3. linked structure

1.Fixed length structure

‘C program printing two’

- In fixed length storage each line of a print is viewed as records when each record have same length.
- Each record has the same length means each record accommodate the same number of characters.

Advantage

- It is used to Express data from record

Disadvantage

- Time is wasted for reading an entire record as most of the storage consists of blank spaces.
- Certain records may require more spaces.

2. variable length structure

This storage variable can be done in two ways.

1 . **Method 1-**

- One can use markers such as two dollar($$) symbol to identify the end of the string.

2. **Method 2-**

One can list the length of string as an identifier in the point for array.

Linked Storage-:

- By a linked list a linear sequence of memory called node where each node contain an item which points to the next node in the list address of the next node is stored.

2.2 CHARACTER DATA TYPE:-

The character data type is of two data type.

(1) Constant
(2) Variable

Constant String:->

The constant string is fixed & is written in either ‘ single quote & — ‖ double quotation.

Ex:-’SONA’

“SONA”

Variable String:

String variable falls into 3 categories.

1.Static
2.Semi-Static
3.Dynamic

1.Static character variable:

Whose variable is defined before the program can be executed & cannot change throughout the program.

2.Semi-static variable:

Whose length variable may as long as the length does not exist, a maximum value. A maximum value determine by the program before the program is executed.

*3.*Dynamic variable:

A variable whose length can change during the execution of the program

2.3 String Operation:

There are four different operations.

1.Sub string

2.Indexing

3.Concatenation

4.Length

1.Sub string:-

- Group of conjunctive elements in a string (such as wards, purchases or sentences) called substring.

  Syntax:: SUBSTRING (String, initial, length)

- To denote the substring of string ‘S‘ beginning in the position ‘K‘ having a length ‘L‘.

For e.g.;

- ❑ SUBSTRING (‘TO BE OR NOT TO BE‘, 4, 7)

  SUBSTRING= BE OR N

- ❑ SUBSTRING (THE END, 4, 4)

  SUBSTRING= END.

2 INDEXING

- Indexing also called pattern matching which refers to finding the position where a string pattern ‘P‘.

- First appears in a given string text ‘T‘, we called this operation index and write as INDEX (text, pattern)
- If the pattern ‘P‘ does not appear in text ‘T‘ then index is assign the value 0;

For e.g.

T contains the text.

‘HIS FATHER IS THE PROFESSOR‘

Then

INDEX (T, ‘THE‘)

7

INDEX (T, ‘THEN‘)

0

3.Concatenation: -

Let S1& S2in be the string then concatenation of S1& S2 is denoted byS1S2, S1||S2, each the string consist of the character of S1followed by the characters ofS2.

Ex:-S1= ‘Sonalisa‘ S2= ‘ ‘ S3= ‘Behera‘

S1||S2||S3= Sonalisa Behera

4.Length operation:-

The number of character in a string is called its length.

We will write LENGTH (string).

For the length of a given string LENGTH (‘Computer’).

The length is 8.

Basic language

LEN (STRING) Strlen (string)

Strupper(string) =Strupr(string) =Strupr(‘computer‘)= COMPUTER

String lower =Strlwr (string) =Strlwr (‘COMPUTER’)= computer

String concatenating =Strcnt
String Reverse =Strrev

Short Questions with answers

Q1. What is string?

String- Finite sequence of zero or more characters is called a string.

- It is denoted by S.
- example- S= “best”

Q2. What is Null String?

string with zero character or no character is known as null string.

- Ex- S=’ ’

Q3. What is Concatenation? [W-2020]

The string consists of the characters of a string S1 followed by the characters of another string S2 is called concatenation of string S1 and S2.

- Concatenation operation of two strings s1 & s2 is denoted by s1||s2 .

- the length of S1||S2= the sum of length of string S1 and S2.
- Example- s1=’BE’ =2 s2=’ST’=2 s1||s2=’BEST’=4

Q4. What is Substring? [S-2015]

A string y is called a substring of s. If there exist string xz such that x||y ||z .

Long Questions

Q1. Explain various string storing methods? [S-2020]

Q2. Explain all the operations of string? [W-2016,2017,2018,2020]

CHAPTER 3
CH-3 (ARRAY)

Articles to be covered

3.1 Give Introduction about array,

3.2 Discuss Linear arrays, representation of linear array in memory

3.3 Explain traversing linear arrays, inserting & deleting elements

3.4 Discuss multidimensional arrays, representation of two-dimensional arrays in memory (row major order & column major order), and pointers

3.5 Explain sparse matrices.

3.1 Introduction

★ Array is a linear homogeneous *data Structure* whose elements are stored in Contiguous memory location .

★ Elements of array are accessed by using index no.

★ Each location of an element in an array has a numerical index which used to identify array

How to declare an array

Syntax

data type variable name [size];

Ex:- Int arr [6];

How to initialize an array

Syntax

datatype variable [size] = {value1, value2,...};

8 };

3.2 Linear array representation in memory :

ARR [K] = base address + (W*K)

K= Index or Position, W= No. of bytes for an element

e.g ARR[4]= 1000+ (2*4) =1008

3.3 Traversing Linear array, inserting & deleting elements

Traversing

❖ In traversing operation of an array, each element of an array is accessed exactly for once for processing.

❖ This is also called visiting of an array.

Here LA is a linear array with lower bound LB and upper bound UB. This algorithm traverses LA applying an operation PROCESS to each element of LA.

Algorithm for Traversing an linear array

1. [Initialize counter] Set k: =LB.
2. Repeat steps 3 and 4 while k <=UB.
3. [Visit Element] Apply PROCESS to LA [k].
4. [Increase Counter] Set k: =k + 1.

[End of step 2 loop]

5. Exit.

LA

A	D	E	Z	K	O
[0]	[1]	[2]	[3]	[4]	[5]

LB = 0
UB= 5

Inserting operation : Inserting refers to a new element into an array

There are 3 types of Insert Operation in array

- Inserting a new element of an array at the beginning
- Inserting a new element of an array at the middle
- Inserting a new element of an array at the ending

1. Inserting a new element of an array at the beginning

Before insertion

2	**18**	**55**	**0**	
[0]	**[1]**	**[2]**	**[3]**	**[4]**

We want to add a new array, at the beginning element "5" to the of this array

If the array is already full then Overflow Condition.

Algorithm

LB=0,UB=3, Size=5

1. START
2. Read LB &UB
3. if UB== Size -1, then OVER FLOW
4 STOP & Exist
5. Else Read Data
6. K = UB
7. Repeat Step 5 to Step 9
8. A [k+1] = A[k]
9. K=K-1
10. A[LB]=Data
11. END

After Insertion

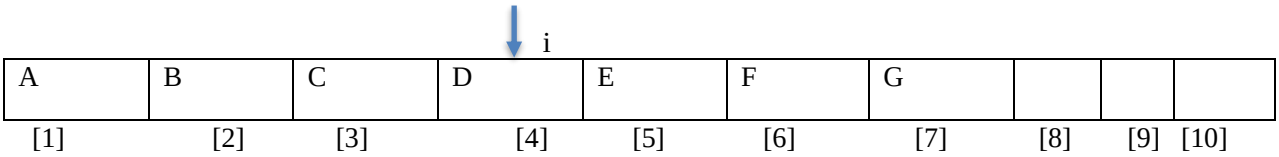
5	2	18	55	0
[0]	[1]	[2]	[3]	[4]

Insert a new element at the end of array

ALGORITHM **LB=0,UB=3,size=5**

1. START
2. Read UB, LB, data
3. IF (UB = = Size -1) then OVERFLOW
4. Exist
5. UB = UB +1
6. A [UB] = data
7. END

Insert a new element at required position in an array

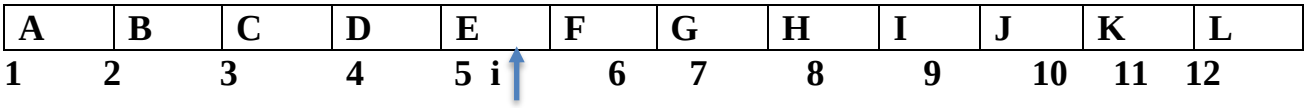


ALGORITHM

N=10,M=7,i=4

1. If(M<N)Then
   Back=M+1
2. Else STOP & EXIT
3. while(Back > i) repeat Step 3 to 4
4. Reg[Back]=Reg[Back-1]
5. Back = Back-1
6. Reg[Back]=data
7. M=M+1
8. End

Deletion Of an element at any position from an array



M= Last element in the array is at Mth position=12

Algorithm

N=12,M=12,i=5

1. Back = i
2. while(Back<M) repeat Step 3 and 4
3. Reg[Back]=Reg[Back+1]
5. Back = Back+1
6. M=M-1
7. End

3.4 Multidimensional Array

1. Array having more than one subscript variable is called Multi-Dimensional array.

2. Multi Dimensional Array is also called as Matrix.

Consider the Two dimensional array -
1. Two Dimensional Array requires Two Subscript Variables
2. Two Dimensional Array stores the values in the form of matrix.
3. One Subscript Variable denotes the Row of a matrix.
4. Another Subscript Variable denotes the Column of a matrix.

Declaration and Use of Two Dimensional Array :

```
int a[3][4];
Use :
for(i=0;i<row;i++)
for(j=0;j<col;j++)
{
printf("%d",a[i][j]);
}
```

Array Representation:

Arrays may be represented in Row-major form or Column-major form.

- Column-major
- Row-major

for example Arr[j,k] of a 2 d array Arr[m,n] can be calculated by using the following formula :-

For example Arr(25,4) is an array with base value 200.w=4 for this array.

The address of Arr(12,3) can be calculated

Row-major Order(RMO)

In Row Major Form, all the elements of the first row are printed, then the elements of the second row and so on up to the last row.
Output:
Row Major:
1 2 3
4 5 6
7 8 9

(Row major order) Address(Arr[j,k])=base(Arr)+w[n(j-1)+(k-1)]

using row-major order as

Address(Arr(12,3))=200+4[4(12-1)+(3-1)]
=200+4[4*11+2]
=200+4[44+2]
=200+4[46]
=200+184
=384

Column Major Order(CMO)

In Column-major form, all the elements of the first column are printed, then the elements of the second column and so on up to the last column.

The ‘C‘ program to input an array of order m x n and print the array contents in row major and column major is given below.

Output
1 4 7
2 5 8
3 6 9

(Column major order) Address(Arr[j,k])= base(Arr)+w[m(k-1)+(j-1)]

Address(Arr(12,3))=200+4[25(3-1)+(12-1)]
=200+4[25*2+11]
=200+4[50+11]
=200+4[61]
=200+244
=444

3.5 Sparse matrix

Matrix with relatively a high proportion of zero entries are called sparse matrix.

Possible Short Questions with answers

Q1. What is array? [2018(W)]

➢ Array is a linear homogeneous data Structure whose elements are stored in Contiguous memory location .

➢ Elements of array are accessed by using index no.

➢ Each location of an element in an array has a numerical index which used to identify array

Q2. What is Traversing operation?

➢ In traversing operation of an array, each element of an array is accessed exactly for once for processing.

➢ This is also called visiting of an array.

Q3. What is Sparse matrix? [2014(W),2017(W),2020(W)]

Matrix with relatively a high proportion of zero entries are called sparse matrix.

Long Questions

Q1. Explain traverse operation on array with algorithm? [2016(W)]

Q2. Explain insert operation on array with algorithm? [2017(W)]

Q3. Describe RMO & CMO?[2020(W)]

CH-4 Stack & Queue

Articles to be covered

4.1 Give fundamental idea about Stacks and queues

4.2 Explain array representation of Stack

4.3 Explain arithmetic expression, polish notation & Conversion

4.4 Discuss application of stack, recursion

4.5 Discuss queues, circular queue, priority queues.

4.1 fundamental idea about Stack

● A stack is a particular kind of abstract data type or collection in which the principal (or only) operations on the collection are the addition of an entity to the collection, known as push and removal of an entity, known as pop.

● The relation between the push and pop operations is such that the stack is a Last-In-First-Out (LIFO) data structure.

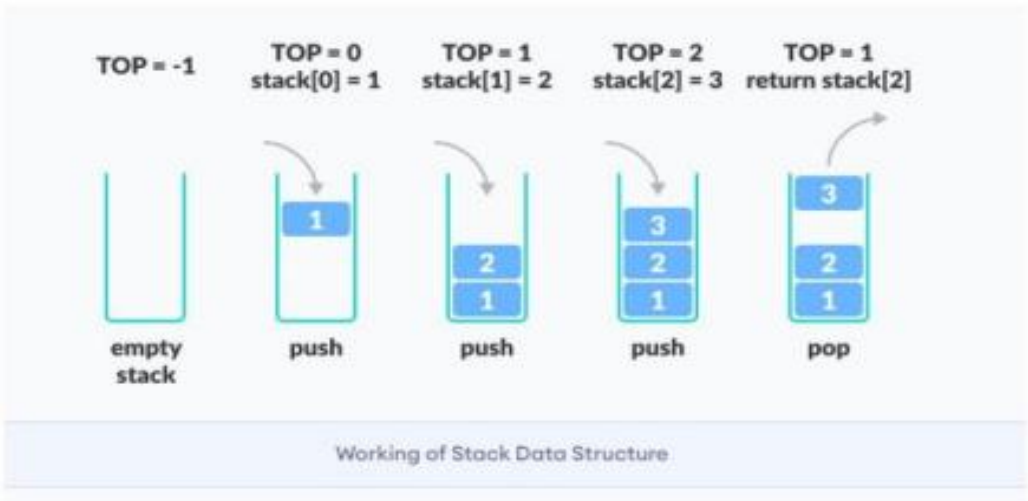
● In a LIFO data structure, the last element added to the structure must be the first one to be removed.

● The push and pop operations occur only at one end of the structure, referred to as the top of the stack.

● If the stack is full and does not contain enough space to accept an entity to be pushed, the stack is then considered to be in an overflow state.

● The pop operation removes an item from the top of the stack.

● A pop either reveals previously concealed items or results in an empty stack, but, if the stack is empty, it goes into underflow state, which means no items are present in stack to be removed.



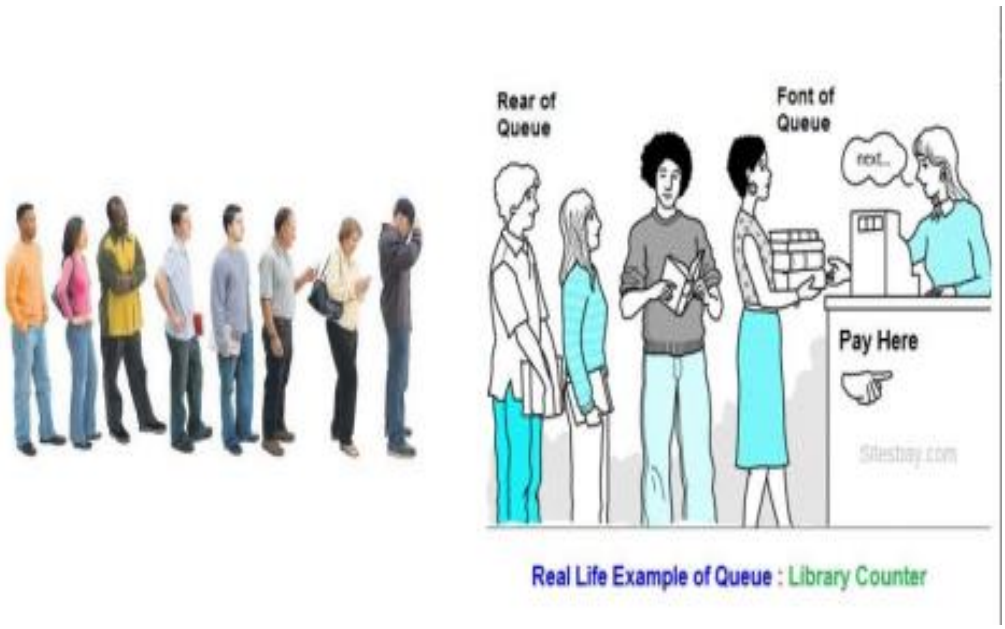
Queues:

Queue is a linear list of elements in which deletions can take place only at one end, called the front and insertions can take place only at the other end, called the rear. The terms"**fron**t" and **"rear"** are used in describing a linear list only

Queue are also called **first-in first-out (FIFO)** lists, since the first elements enter a queue will be the first element out of the queue.

In other words, the order in which elements enter a queue is the order in which they leave. This contrasts with stacks, which are last-in first-out (LIFO) lists.

Queues abound in everyday life. The automobiles waiting to pass through an intersection form a queue. In which the first car in line is the first car through; the people waiting in line at a bank form a queue, where the first person in line is the first person to be waited on; and so on. An important example of a queue in



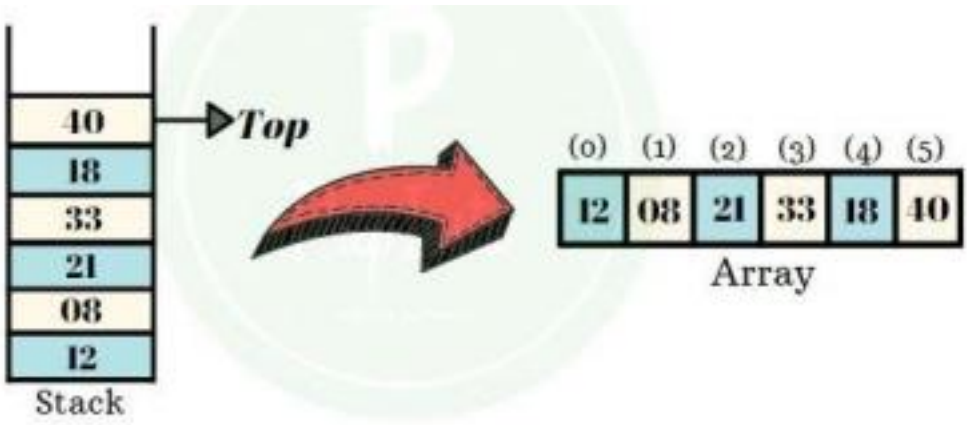
4.2 Array representation of Stack

In most high level languages, a stack can be easily implemented either through an array or a linked list.

Array

The array implementation aims to create an array where the first element(usually at the zero-offset) is the bottom. That is, array[0] is the first element pushed onto the stack

and the last element popped off. The program must keep track of the size, or the length of the stack.



Algorithm 1: PUSH (STACK, TOP, MAXSTK, ITEM)

This procedure pushes an item on to a stack.

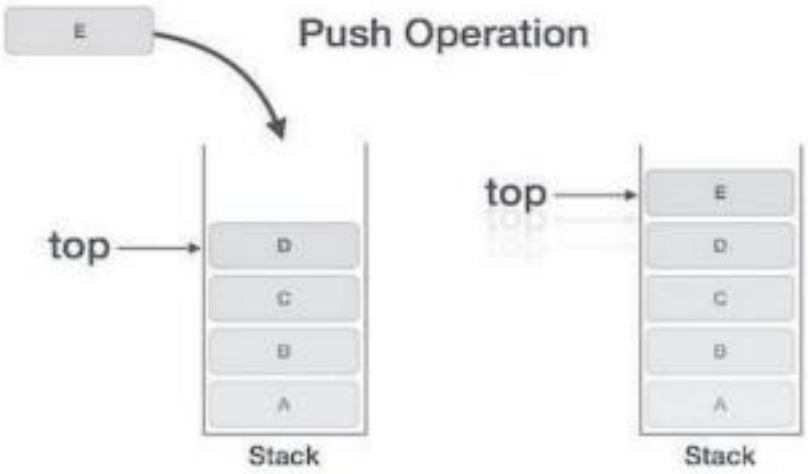
1. [Stack already filled]?

   If TOP = MAXSTK, then: Print: OVERFLOW, and Return.

2. Set TOP: = TOP + 1. [Increase TOP by 1].

3. Set STACK [TOP]: = ITEM. [Inserts ITEM in new TOP position].

4. Return.



Algorithm 2: POP (STACK, TOP, ITEM)

This procedure deletes the TOP element of STACK and assigns it to the variable ITEM.

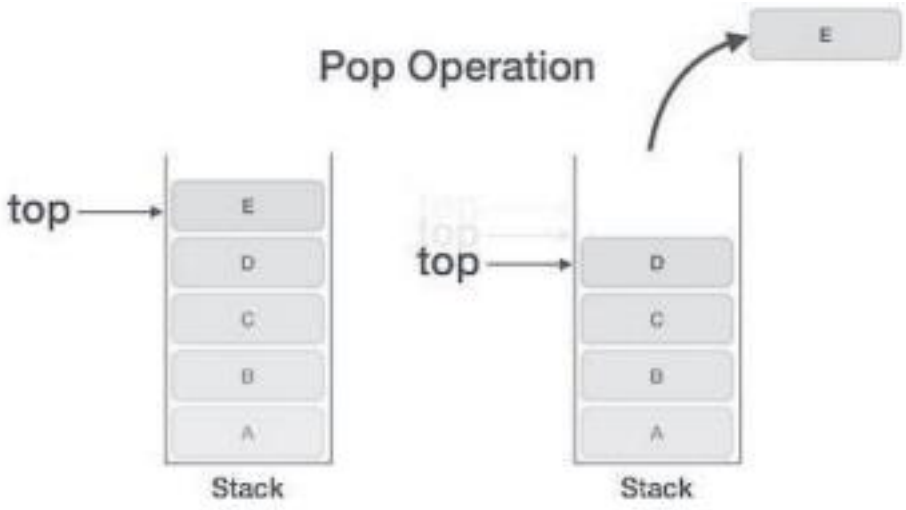
1. [Stack has an item to be removed]

   If TOP = 0, then: Print: UNDERFLOW and Return.

2. Set ITEM: =STACK [TOP]. [Assign TOP element to ITEM].

3. Set TOP: = TOP – 1 [Decrease TOP by 1].

4. Return.



4.3 Arithmetic expression, polish notation & Conversion Polish notation

● Polish notation, also known as Polish prefix notation It places operators to the left of their operands.
● The term Polish notation is sometimes taken (as the opposite of infix notation) to also include Polish postfix notation, or Reverse Polish notation, in which the operator is placed after the operands .

Conversion between Infix, Prefix and Postfix Notation

Infix	Prefix	Postfix
a + b	+ a b	a b +
a + b * c	+ a * b c	a b c * +
(a + b) * (c - d)	* + a b - c d	a b + c d - *
b * b - 4 * a * c		
40 - 3 * 5 + 1		

Postfix expressions are easily evaluated with the aid of a stack.

Conversion from INFIX to POSTFIX

Algorithm Polish (Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

S-1: If the scanned character is an operand, put it into the Postfix Expression

S-2: If the scanned character is an operator and stack is empty, push it into the stack.

S-3: If stack is not empty, then there may be following possibilities

a) If the precedence of scanned operator is greater than the top most operator, push this scanned operator into stack.
b) If the precedence of scanned operator is less than the top most operator, pop the operators from stack until we find a low/equal precedence operator than scanned character.
c) If the precedence of scanned operator is equal then check the associativity of the operator

    i) If the associativity is (L-> R) then pop the operators from stack until we find a low precedence operator
    ii) If the associativity is (R-> L) then simply put operator into stack.
d) If the scanned operator is opening bracket '(' ,then push into Stack.
e) If the scanned operator is closing bracket ')' ,then pop out operators from stack until we find an opening bracket '('.

Repeat Step 1,2,3 till expression is ended.

S-4: Pop out all the remaining operators from stack and push them into Postfix Expresssion.

S-5: Exit

PRECEDENCE CHART FOR OPERATORS

1. ( )

2. ^ (RIGHT->LEFT)

3. * / (LEFT -> RIGHT)

4. + - (LEFT ->RIGHT)

Conversion from INFIX to PREFIX

Algorithm Polish (Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent prefix expression P.

S-1: First reverse the given Expression

S-2: If the scanned character is an operand, put it into the Prefix Expression

S-2: If the scanned character is an operator and stack is empty, push it into the stack.

S-3: If stack is not empty, then there may be following possibilities

a) If the precedence of scanned operator is greater than the top most operator, push this scanned operator into stack.

b) If the precedence of scanned operator is less than the top most operator, pop the operators from stack until we find a low/equal precedence operator than scanned character.

c) If the precedence of scanned operator is equal then check the associativity of the operator

i. If the associativity is (R->L) then pop the operators from stack until we find a low precedence operator
ii. If the associativity is (L->R) then simply put operator into stack.
iii.

d) If the scanned operator is closing bracket ')' ,then push into Stack.

e) If the scanned operator is opening bracket '(' ,then pop out operators from stack until we find an closing bracket ')'.

Repeat Step 2,3,4 till expression is ended.

S-4: Pop out all the remaining operators from stack and push them into Prefix Expression.

S-5: Reverse the expression

S-6: Exit

4.4 Application of stack, recursion

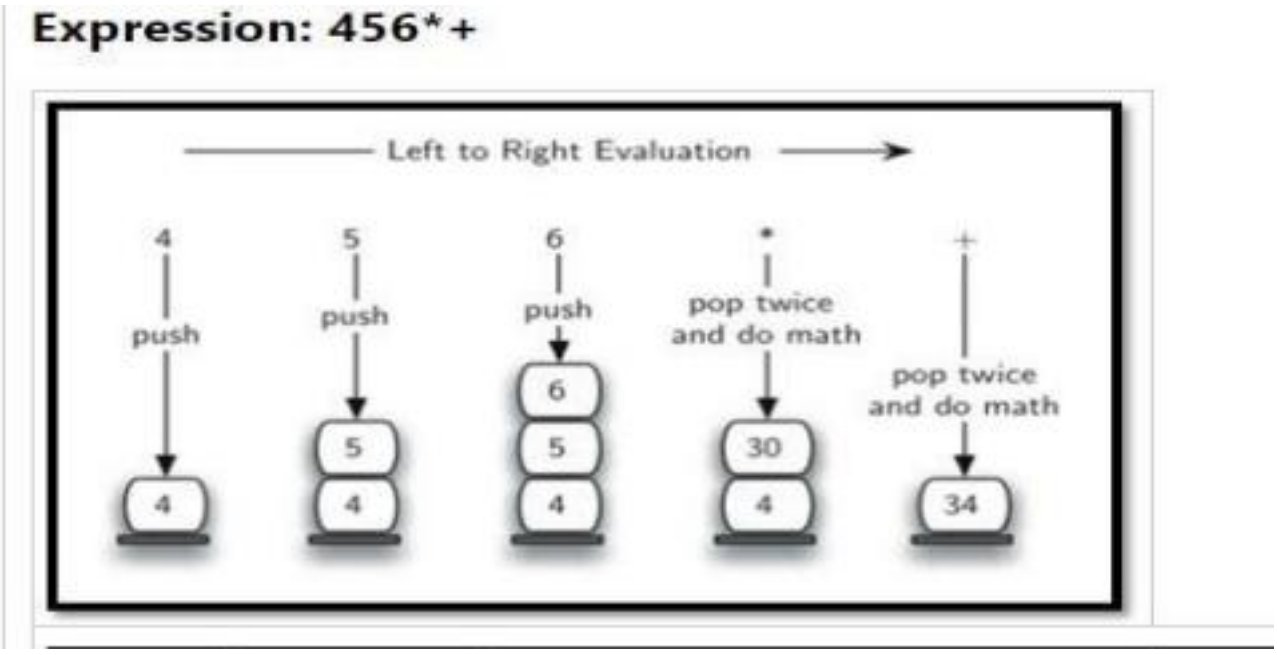
➢ **Recursion** is the process of repeating items in a self-similar way.

➢ A classic example of recursion is the definition of the factorial function, given here in C code:

```
unsigned int factorial(unsigned int n) {
if (n == 0) {
return 1;
} else {
return n * factorial(n - 1);
}
```

Algorithm: This algorithm finds the VALUE of an arithmetic expression P written in postfix notation.

1. Add the right parentheses‖)‖ at the end of P.
2. Scan P from left to right and repeat step 3 and 4 for each element of P until the sentinel) is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator is encountered, then:
a. Remove the two top elements from the STACK
b. Evaluates these two operators using that operator.
c. Place the result of (b) back on STACK.
[End of If structure]
[End of step 2 loop]
5. Set VALUE equal to the top element of STACK.
6. EXIT.



Step	Input Symbol	Operation	Stack	Calculation
1.	4	Push	4	
2.	5	Push	4,5	
3.	6	Push	4,5,6	
4.	*	Pop(2 elements) & Evaluate	4	5*6=30
5.		Push result(30)	4,30	
6.	+	Pop(2 elements) & Evaluate	Empty	4+30=34
7.		Push result(34)	34	
8.		No-more elements(pop)	Empty	34(Result)

4.5 Discuss queues, circular queue, priority queues.

QUEUE

Queue is a linear list of elements in which deletions can take place only at one end, called the front and insertions can take place only at the other end, called the rear.

The terms **"front"** and **"rear"** are used in describing a linear list only when it is implemented as a queue.

Queue are also called **first-in first-out** (FIFO) lists, since the first elements enter a queue will be the first element out of the queue. In other words, the order in which elements enter a queue is the order in which they leave.

Representation of queues:

Queues may be represented in the computer in various ways, usually by means at one way lists or linear arrays. Unless otherwise stated or implied, each of our queues will be maintained by a linear array QUEUE and two pointer variables:

• FRONT, containing the location of the front element of the queue

• REAR, containing the location of the rear element of the queue.

Operations Implemented over Queue

1. ENQUEUE(DATA)
2. DEQUEUE

ENQUEUE : Insert new element into Queue.

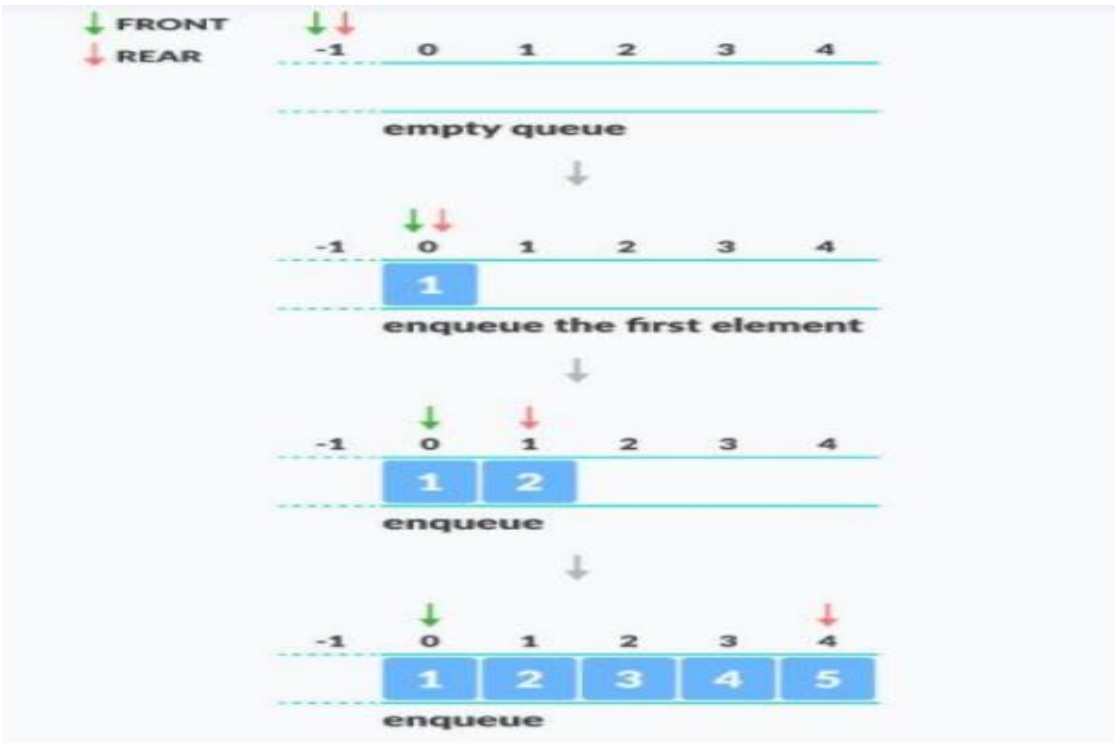
Algorithm For ENQUEUE(data)

S-1: Check if the **queue is full**.
if (rear== size-1) Then
S-2: OVERFLOW condition. STOP & EXIT
S-3: Else Check the Queue is Empty.
If (front ==-1 and rear == -1) Then
Set front and rear pointer to 0 Value & put item into the rear position.
S-4: Else Increment rear pointer to point the next space
S-5: Add data element to the Queue location where the rear pointer is pointing
S-6: EXIT

Algorithm For DEQUEUE

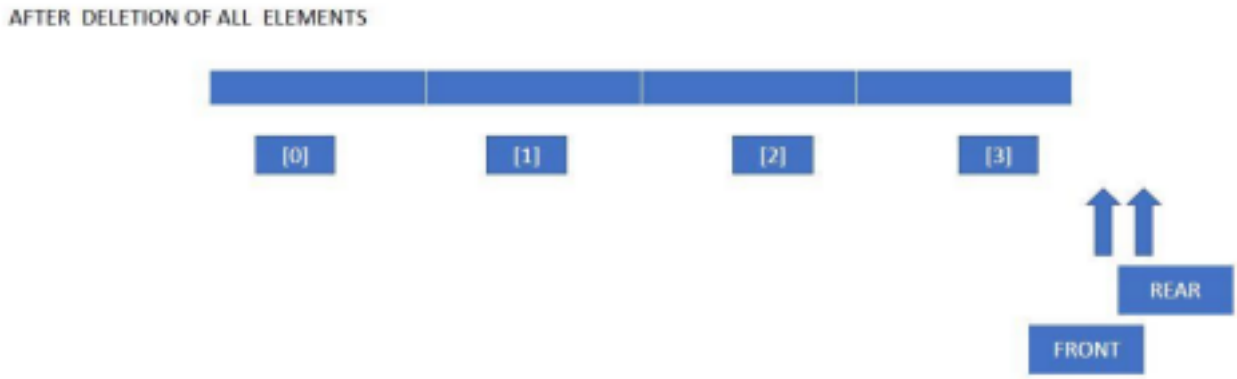
S-1: Check if the **queue is Empty**.
if (front = = -1 AND rear = =-1) Then
S-2: UNDERFLOW condition. STOP & EXIT
S-3: Else Check the Queue has only one element.
If (front = = rear) Then

Set front and rear pointer to -1 Value & delete the element
S-4: Access the data where front is pointing
S-5: Else Increment front pointer to point the next available data
S-6: EXIT



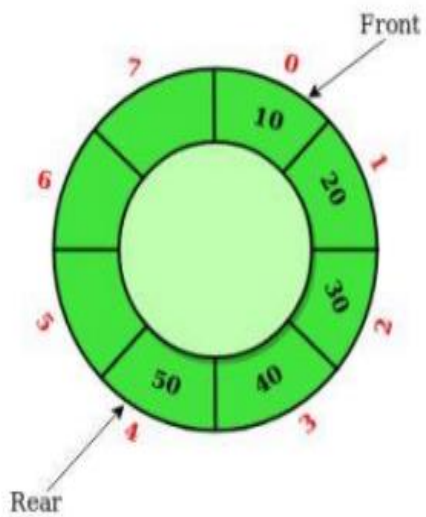
Disadvantage of Linear/Static Queue

When any element is inserted in linear queue then rear will be increased by 1. Let, assume after insertion operations rear is shifted to last position in queue. It means, now queue is full. Now if a new element is inserted then overflow condition will occur.



Circular queue

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called 'Ring Buffer'.

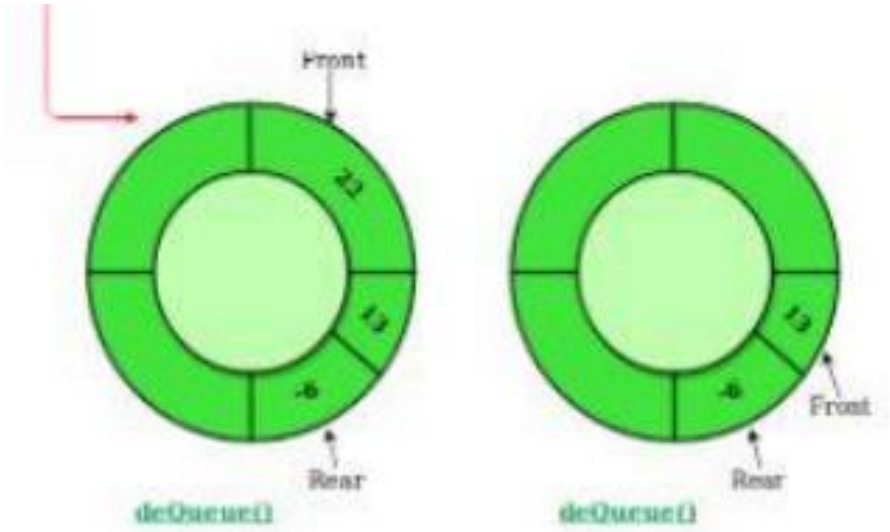


Enqueue Operation

S-1: Check if the Circular **Queue is full**.
If (((rear+1)%N)==front) Then
S-2: Overflow Error, STOP & EXIT
S-3. Else Check Circular **Queue is Empty**
If(front = rear = -1)
S-4: Set value of both front & rear to 0 AND Put data element to rear pointer pointing location.
S-5: Else Increase pointer value as rear=((rear+1)%N)
S-6: Put data element to rear pointer pointing location.
S-7: EXIT

Dequeue Operation

S-1: Check Circular Queue is EMPTY
If (front = rear = -1) Then
S-2: UNDERFLOW ERROR, STOP & EXIT
S-3: ELSEIF Check only one element is present in Circular Queue
If (front=rear) Then
Set both front AND rear value to -1
S-4: Access the data where front is pointing
S-5: ELSE Increase front pointer as (front=(front+1)%N)
S-6: EXIT



Priority Queues:

- A priority queue is a collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the which following rule:
    - An element of higher priority is processes before any element of lower priority.
    - Two elements with the same priority are processes according to the order in, which they were added to the queue.
- A prototype of a priority queue is a timesharing system: programs of high priority are processed first, and programs with the same priority form a standard queue.
- There are various ways of maintaining a priority queue in memory.

Possible Short Questions with Answer

Q1. What is Stack? [W-2018,2019]

- ➢ A stack is a particular kind of abstract data type or collection in which the principal (or only) operations on the collection are the addition of an entity to the collection, known as push and removal of an entity, known as pop.

- ➢ The relation between the push and pop operations is such that the stack is a Last-In-First-Out (LIFO) data structure

Q2. What is queue? [S-2018]

- ➢ Queue is a linear list of elements in which deletions can take place only at one end, called the front and insertions can take place only at the other end, called the rear.
- ➢ The terms "front" and "rear" are used in describing a linear list only when it is implemented as a queue.
- ➢ Queue are also called first-in first-out (FIFO) lists, since the first elements enter a queue will be the first element out of the queue. In other words, the order in which elements enter a queue is the order in which they leave.

Q3.What is polish notation? [W-2014]

- ➢ Polish notation, also known as Polish prefix notation It places operators to the left of their operands.
- ➢ The term Polish notation is sometimes taken (as the opposite of infix notation) to also include Polish postfix notation, or Reverse Polish notation, in which the operator is placed after the operands .

Long Questions

Q1. Explain PUSH & POP algorithm on Stack? [W-2014,2015,2017,2020]

Q2. Explain Enqueue & Dequeue operation on Queue? [W-2015,2016]

Q3. Define Queue? Explain the overflow and underflow condition of queue.[W-2019]

Q4. Write algorithm for finding the VALUE of an arithmetic expression P written in postfix notation.?[W-2018,2019]

CHAPTER 5
LINKED LIST

Articles to be covered

5.1 Give Introduction about linked list

5.2 Explain representation of linked list in memory

5.3 Discuss traversing a linked list, searching,

5.4 Discuss garbage collection.

5.5 Explain Insertion into a linked list, Deletion from a linked list, header linked list

5.1 Introduction about linked list

The linked list is a linear data structure where each node has two parts.

1. Data
2. Reference to the next node

1. DATA

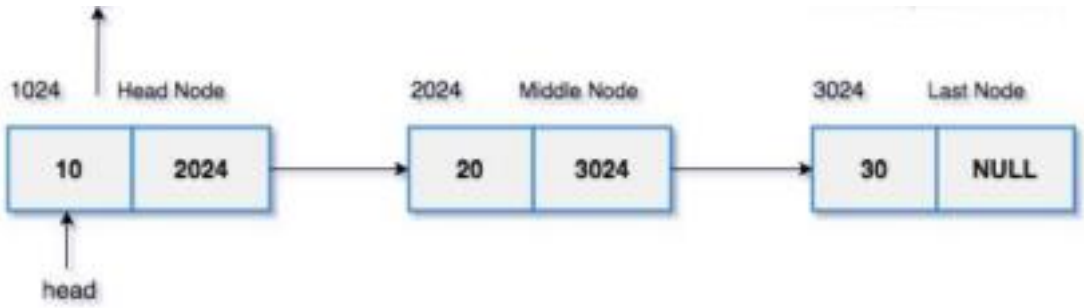
we can store the required information. It can be any data type.

2. Reference to the next node.

It will hold the next nodes address.

Head Node - Starting node of a linked list.

Last Node - Node with reference pointer as NULL.



1. head => next = middle. Hence head => next holds the memory address of th middle node (2024).

2. middle => next = last. Hence middle => next holds the memory address of the last node (3024).

3. last => next = NULL which indicates it is the last node in the linked list.

Advantages of linked list

➢ Dynamic data structure that can grow an string. Efficient memory utilization (exact amount of data storage). Insertion deletion & pupation are easy & efficient.

➢ Data stored in RAM but not sequential .

Disadvantages of linked list

➢ More memory space is needed if no. of filed are more. Logical & physical ordering of node are different. Searching is solve .
➢ Difficult to program because pointer manipulation is required.

Types of linked list:-

➢ Linear linked list or one way linked list or single list.
➢ Double linked list or two way linked list are two way linked list.
➢ Circular linked list is two types i.e.

(1) Single circular list
(2) Double circular list.

Linear linked list:-

➢ It is a one way collection of nodes where the linear order is maintained by pointers.
➢ Nodes are not in sequence, each node implemented by a self referential structure
➢ Each node is divided in two parts.

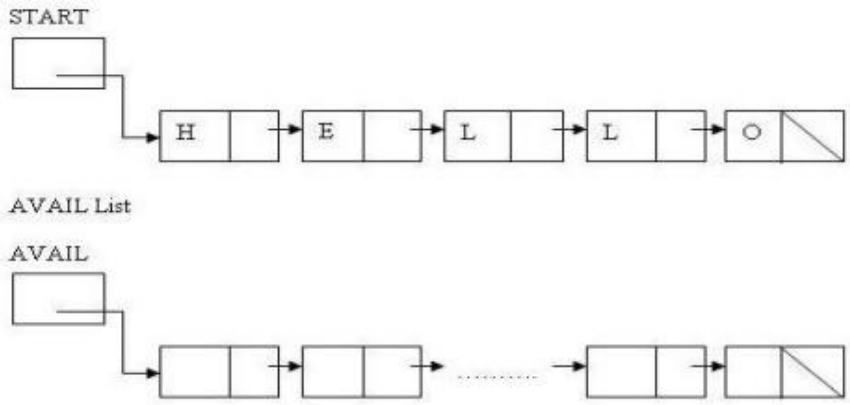
- First part contain the information of the element (INFO).
- Second part is linked field contains the add. Of next node in the list (LINK) field or next pointer filed .

5.2 Memory allocation of the linear linked list:-

- The computer maintains a special list which consist of a list of all free memory calls & also has its own pointer is called the list of available space or the free storage list or the free pool .

- Suppose insertion are to be performed on linked list then unused memory calls the array will also be linked to gather to form a linked list using AVAIL.
- As its list pointer variable such a data structure will be denoted by writing

LIST(INFO,LINK,START,AVAIL)



Create a node

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

struct  node *head,*newNode;

newNode=(struct node*)malloc(sizeof(struct node))
```

5.3 Discuss traversing a linked list, searching

TRAVERSE LINKLIST -: To print each node's data, we have to traverse the linked list till the end.

ALGORITHM

Step-1: Check if linked list is Empty, Then

If(head==NULL)

Step-2: Linked List is Empty, nothing to traverse

Step-3: ELSE Create a temporary node(temp) and assign the head node's address.

temp=(struct node*)malloc(sizeof(struct node))

temp=head

Step-4: Print the data which present in the temp node.

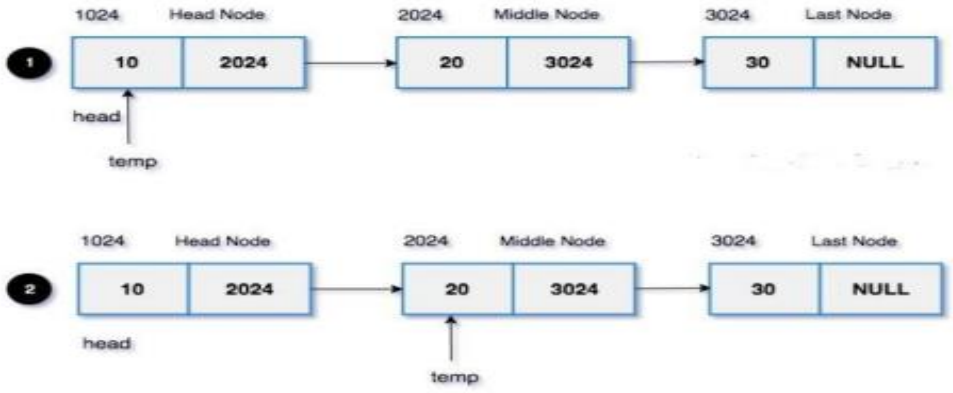
printf(temp->data)

Step-5: After printing the data, move the temp pointer to the next node.

temp=temp->next

Repeat the Step-4 & 5 till temp->next!=NULL.

Step-6: EXIT



SEARCHING LINKED LIST

Check whether the given key is present or not in the linked list.

ALGORITHM

Step-1: Check if linked list is Empty, Then

If(head==NULL)

Step-2: Linked List is Empty, nothing to traverse

Step-3: ELSE Create a temporary node(temp) and assign the head node's address.

temp=(struct node*)malloc(sizeof(struct node))

temp=head

Step-4:Traverse Linked list (USING LOOP)

Step-5: ELSE check If any node has the given key value, Then

If(temp->data==key)

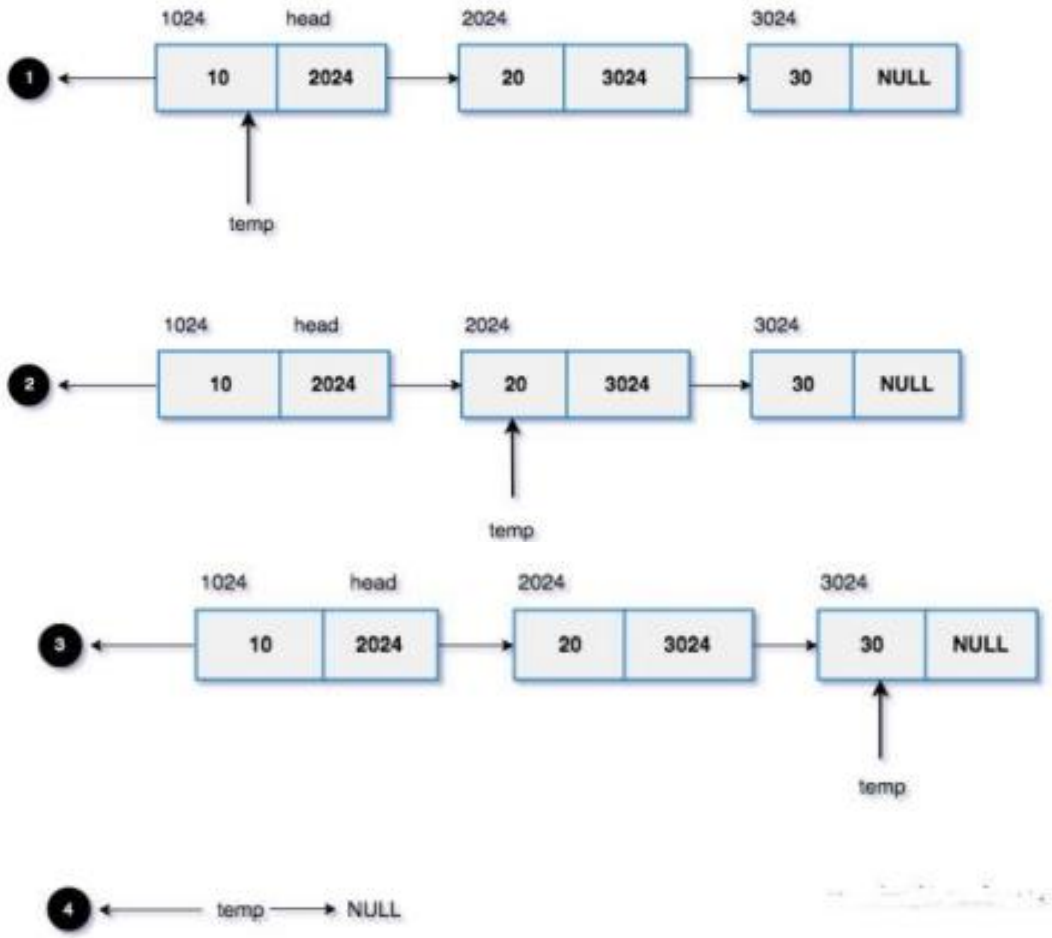
Step-6: Search is Successful

Step-7: ELSE Moving temp=temp->next

Repeat Step-5,6,7 till till temp->next!= NULL

Step-8: If the program execution comes out of the loop (the given key is not present)

Step-9: EXIT



5.4 Discuss garbage collection

➢ The operating system of a computer may periodically collect all the deleted space on to the free storage list. Any technique which does these collections is called garbage collection.

➢ When we delete a particular note from an existing linked list or delete the linked list the space occupied by it must be given back to the free pool. So that the memory can be the used by some other program that needs memory space.

➢ The operating system will perform this operation whenever it finds the CPU idle or whenever the programs are falling short of memory space.

➢ The OS scans through the entire memory cell & marks those cells. That are being by some program then it collects the entire cell which are not being used & add to the free pool.

➢ So that this cells can be used by other programs. This process is called garbage collection. The garbage collection is invisible to the programmer.

5.5 Explain Insertion into a linked list, Deletion from a linked list, header linked list

INSERTION

i) Inserting a node at the beginning of a linked list

Algorithm

Step-1: Create a new node with the given data.
struct node *newNode = malloc(sizeof(struct node));

Step-2: Store the data in the next part of newnode
newNode-> data= val

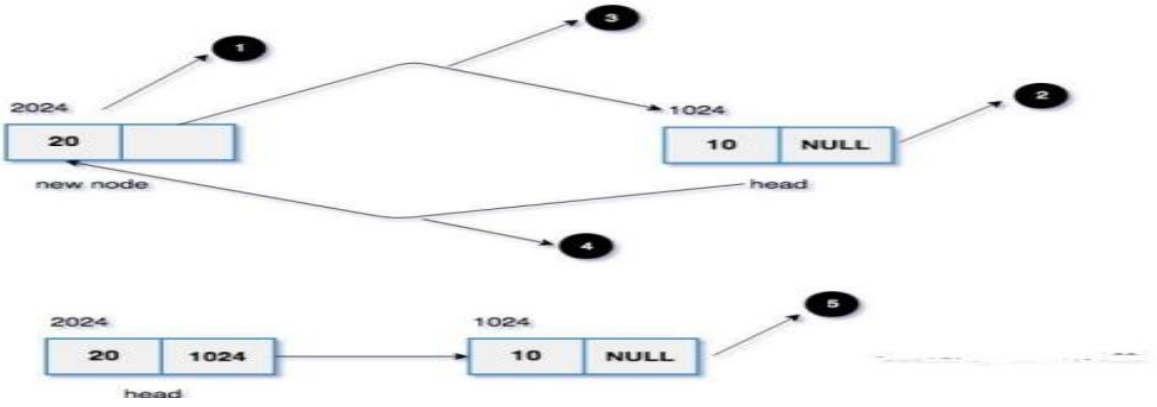
Step-3: Make the new node points to the head node.
newNode->next = head;

Step-4: Finally, make the new node as the head node.

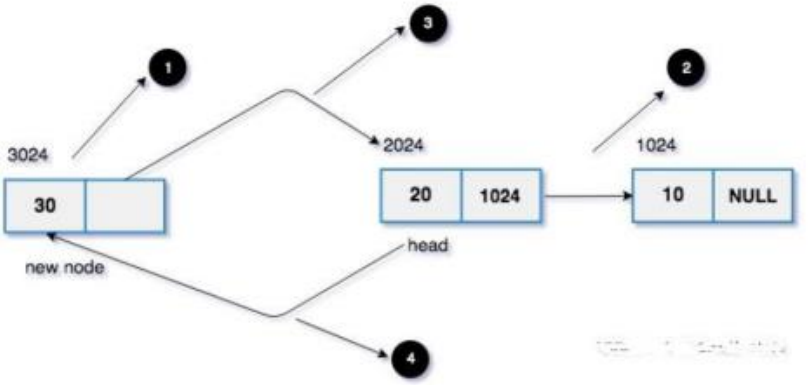
head = newNode;

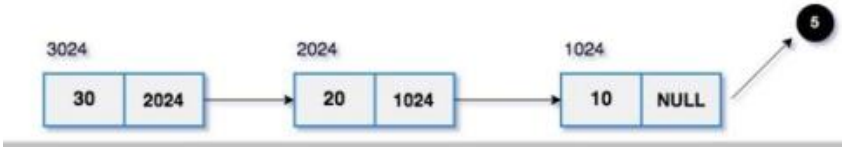
Step-5: EXIT

Let's insert data 20.



Let's insert data 30.





ii) Inserting a node at the end of a linked list

The new node will be added at the end of the linked list.

ALGORITHM

Step-1 : Create a new node with the given data. And make the new node => next as NULL. (Because the new node is going to be the last node.)

```
struct node *newNode = malloc(sizeof(struct node));
newNode->data = val;
newNode->next = NULL;
```

Step-2: Create a temporary node(temp) and assign the head node's address.

temp=head

Step-3: Traverse the linked list (Using LOOP)by temp node till temp->next!, = NULL

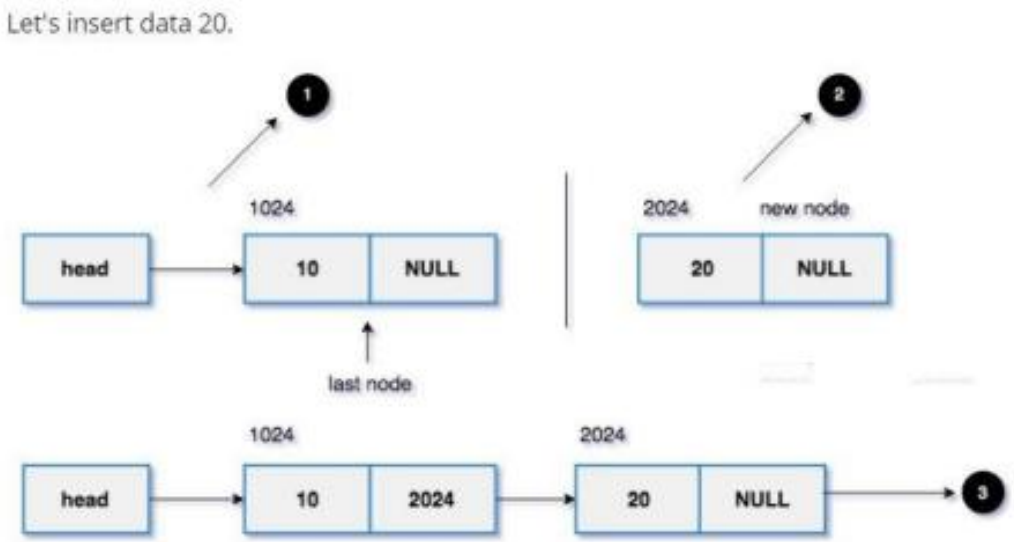
temp= temp->next

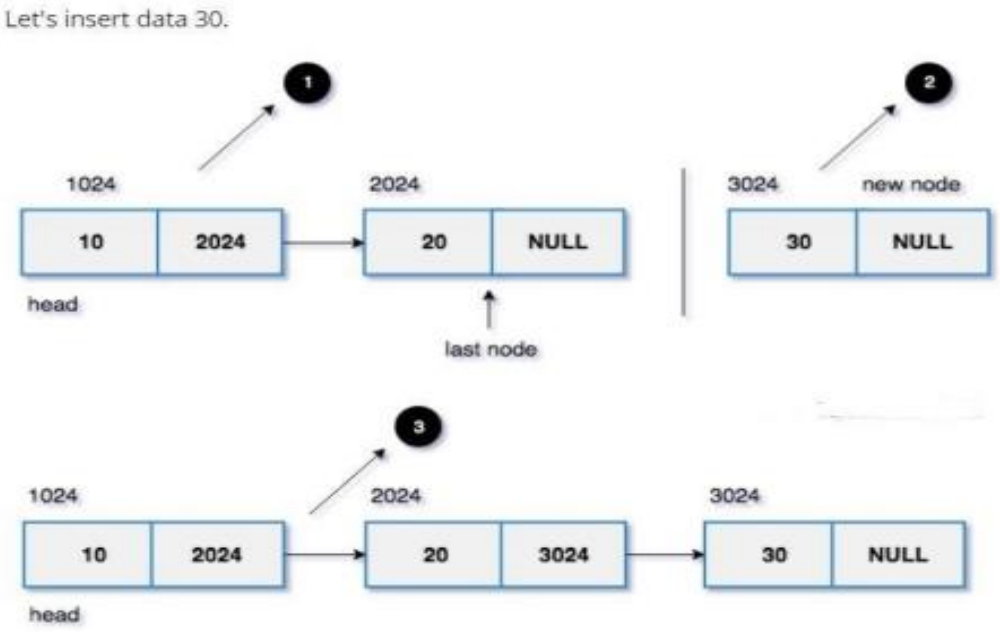
Step-4: Store address of newNode to temp->next

Temp->next=newNode

Step-5: make the newNode->next=NULL

Step-6: EXIT





iii) Insert new element at any position

pos= at which index element is to be inserted,
count=total no. of nodes are present in linked list
i=1

Step-1: Create a new node with the given data. And make the new node => next as NULL. (Because the new node is going to be the last node.)

```
struct node *newNode = malloc(sizeof(struct node));
newNode->data = val;
newNode->next = NULL;
```

Step 2: Check if pos > count ,Then
Step-3: INVALID POSITION ERROR, STOP & EXIT

Step-4: ELSE Create a temporary node(temp) and assign the head node's address.

temp=head

Step-5: Assign temp=temp->next till i<pos ,increase i

Repeat Step-4 till Condition is valid

Step-6 : Make newNode->next=temp->next

Step-7: Make temp->next=newNode

Step-8: EXIT

DELETION:

i) Deleting a node at the beginning of a linked list

Step-1: Check whether there is only one node in the list
if(head->next=NULL)
Step-2: Make head=NULL & free(head)
Step-3: ELSE assign temp=head
Step-4: Make head=head->next
Step-5: free(temp)
Step-6: EXIT

ii) Deleting a node at the end of a linked list

Step-1: Check whether there is only one node in the list
if(head->next=NULL)
Step-2: Make head=NULL & free(head)

Step-3: ELSE Create two nodes as prevNode, temp

Step-4: Assign temp=head

Step-5: Traverse the linked list (Using LOOP)by temp node till temp->next!= NULL

Step-6: Make prevNode=temp & temp= temp->next

Step-7: prevNode->next=NULL & free(temp)

Step-8: EXIT

iii) Deleting a node at any position of a linked list

pos= at which index element is to be inserted,
count=total no. of nodes are present in linked list
i=1
Step-1: Check if pos > count ,Then
Step-2: INVALID POSITION ERROR, STOP & EXIT
Step-3: Check whether there is only one node in the list
if(head->next=NULL)
Step-4: Make head=NULL & free(head)
Step-5: ELSE Create two nodes as nextNode, temp
Step-6: Assign temp=head
Step-7: Move temp=temp->next(USING LOOP)increasing i, till i<pos-1
Step-8: Assign nextNode=temp->next & temp->next=nextNode->next
Step-9: free(nextNode)
Step-10: EXIT

Short Questions

Q1. What is Linked List? [2018(w)]

The linked list is a linear data structure where each node has two parts. 1. Data
2. Reference to the next node

Q2. What are the operation perform on Linked list?

- TRAVERSE
- Searching
- Insert
- Delete

Q3. What is garbage collection? [2011(w),2012(w),2017(w)]

The operating system of a computer may periodically collect all the deleted space on to the free storage list. Any technique which does these collections is called garbage collection.

Long Questions

Q1. Write an algorithm to traverse a linked list? [2014(w),2017(w),2018(w),2019(w),2020(w)]

Q2. Write an algorithm to insert a node at the end of a linked list with a suitable example?[2018(w)]

Chapter 6 Tree

Articles to be covered

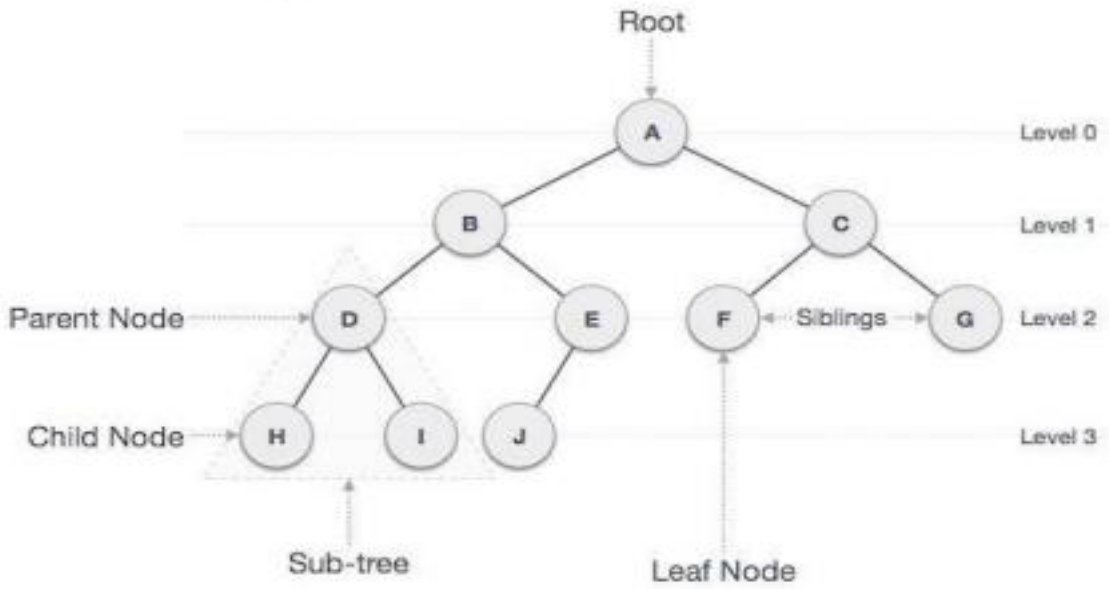
6.1 Explain Basic terminology of Tree

6.2 Discuss Binary tree, its representation and traversal, binary search tree, searching,

6.3 Explain insertion & deletion in a binary search trees

6.1 Explain Basic terminology of Tree

A tree is a non-linear data structure that consists of a root node and potentially many levels of additional nodes that form a hierarchy. A tree can be empty with no nodes called the null or empty tree or a tree is a structure consisting of one node called the root and one or more subtrees.



Thus tree is a finite set of one or more nodes such that :

i> There is a specially designated node called the root

ii> The remaining nodes are partitioned into n >= 0 disjoint sets T1,T2,.....,Tn where each of these sets is a tree . T1,T2,.....,Tn are called the subtrees of the root .

Terminologies used in Trees

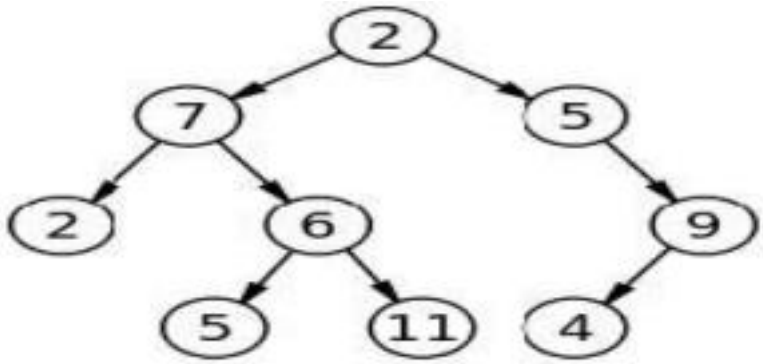
- Root - the top node in a tree.
- Node - the item of information .
- Parent - the converse notion of child.
- Siblings - nodes with the same parent.
- Children nodes – roots of the subtrees of a node , X , are the children of X . • Descendant - a node reachable by repeated proceeding from parent to child. • Ancestor - a node reachable by repeated proceeding from child to parent. • Leaf or Terminal node - a node with no children (degree zero) .
- Nonterminal nodes – nodes other than terminal nodes .
- Internal node - a node with at least one child.
- External node - a node with no children.
- Degree - number of sub trees of a node.

• Edge - connection between one node to another.

• Path - a sequence of nodes and edges connecting a node with a descendant. • Level - The level of a node is defined by 1 + the number of connections between the node and the root.

• Height - The height of a node is the length of the longest downward path between the node and a leaf.

• Forest - A forest is a set of n ≥ 0 disjoint trees. If we remove the root of a tree we get a forest.

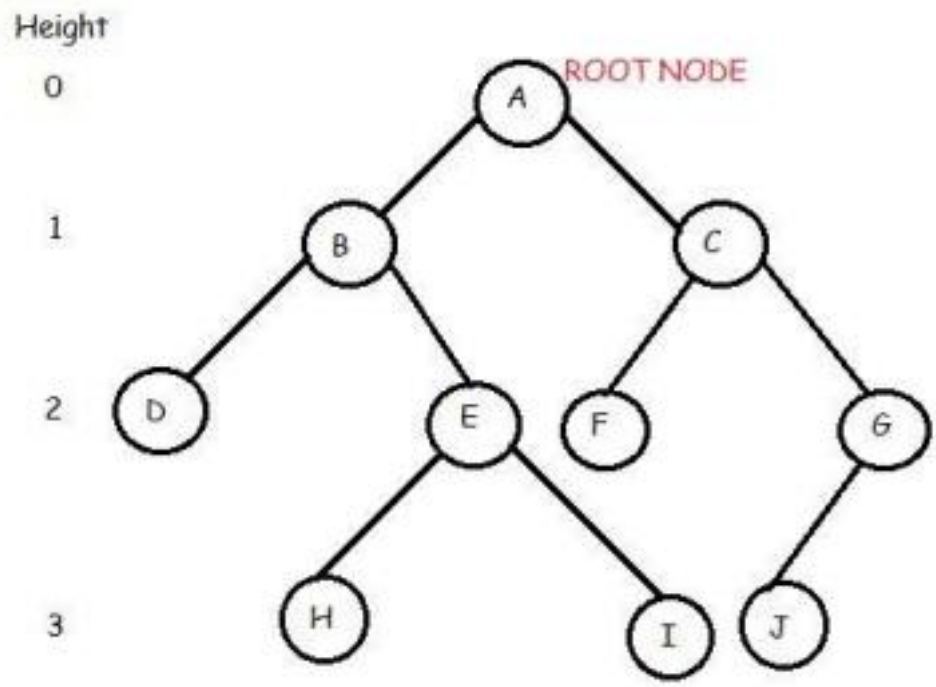


A simple unordered tree

6.2 Discuss Binary tree, its representation and traversal, binary search tree, searching

Binary Tree

A binary tree is a hierarchical data structure in which each node has at most two children generally referred as left child and right child.



In the above binary tree we see that root node is A. The tree has 10 nodes with 5 internal nodes, i.e, A,B,C,E,G and 5 external nodes, i.e, D,F,H,I,J. The height of the tree is 3. B is the parent of D and E while D and E are children of B.

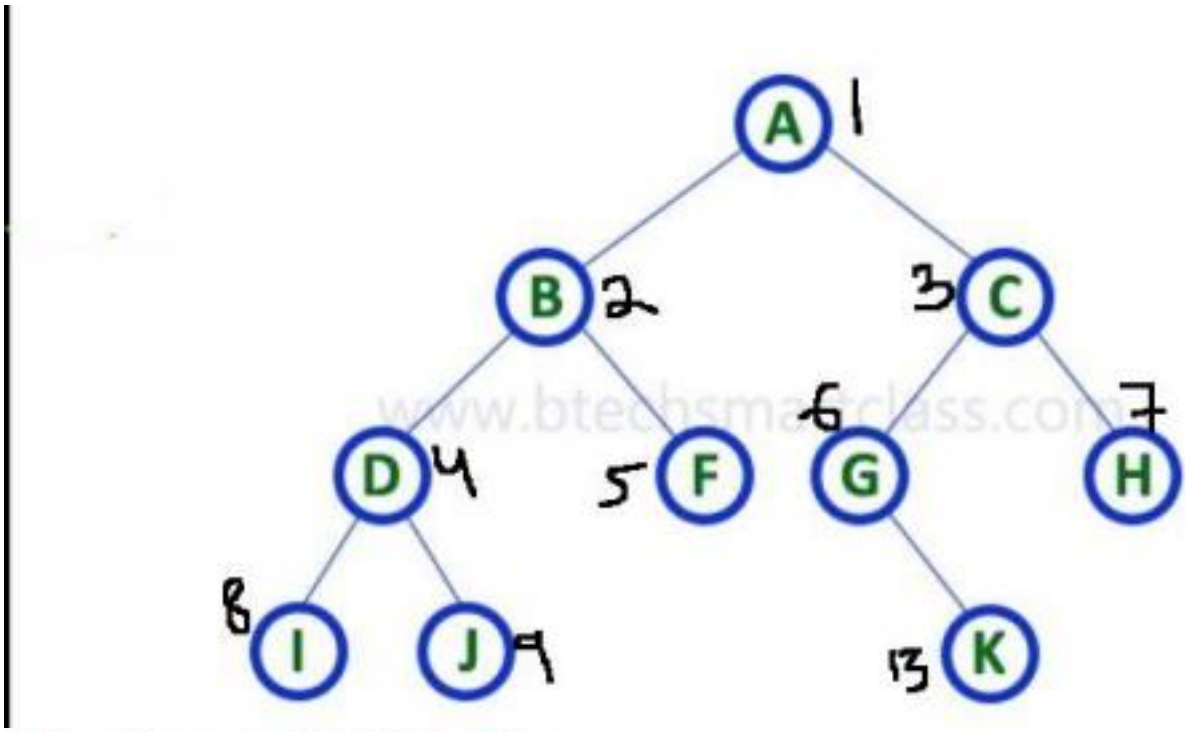
Binary Tree Representations

The nodes may now be stored in a one dimensional array tree, with the node numbered i being stored in tree(i).

(i) parent(i) is at [i/2] if i is not equal to 1. When i=1, i is the root and has no

parent. (ii) lchild(i) is at 2i if 2i <= n . If 2i > n, then i has no left child.

(iii) rchild(i) is at 2i+1 if 2i+1 <= n .If 2i+1 > n, then i has no right child.



1. Array Representation of Binary Tree

In array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.

Consider the above example of a binary tree and it is represented as follows...

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

Binary Tree Traversal

- Inorder
- postorder
- preorder

Algorithm For Inorder Traversal

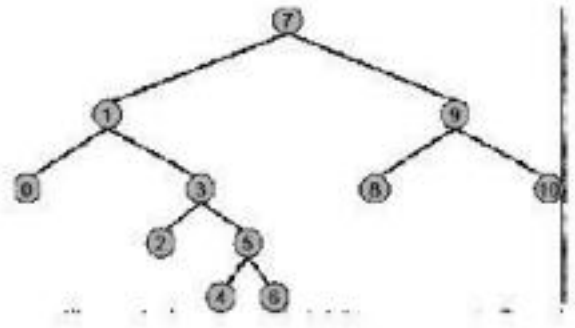
Step-1: Repeat steps 2 and 4 till Tree != NULL
Step-2: INORDER(TREE->LEFT)
Step-3: PRINT(TREE->ROOT)
Step-4: INORDER(TREE->RIGHT)
END OF LOOP
Step-5: EXIT

Algorithm For Preorder Traversal

Step-1: Repeat steps 2 and 4 till Tree != NULL
Step-2: PRINT(TREE->ROOT)
Step-3: PREORDER(TREE->LEFT)
Step-4: PREORDER(TREE->RIGHT)
END OF LOOP
Step-5: EXIT

Algorithm For Postorder Traversal

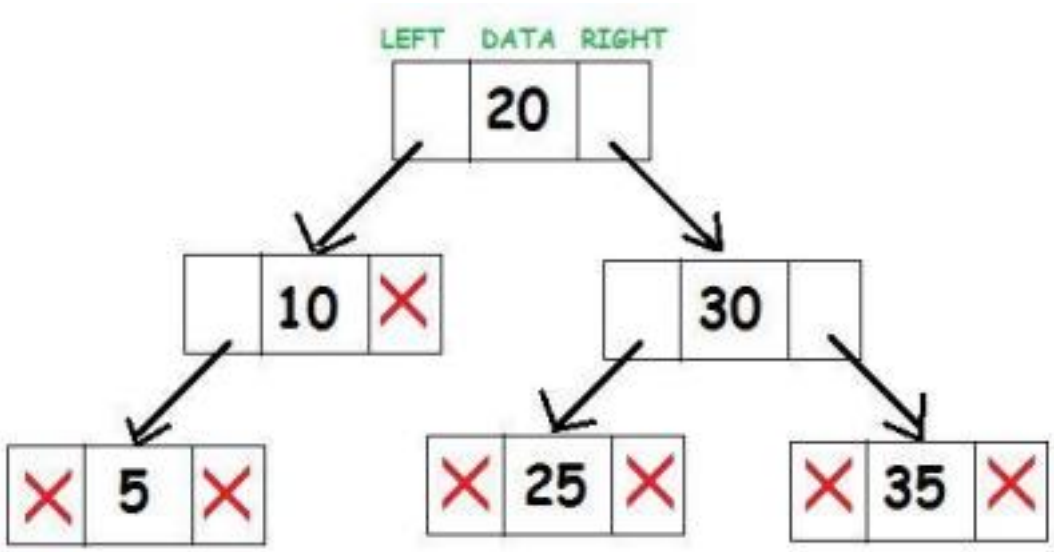
Step-1: Repeat steps 2 and 4 till Tree != NULL
Step-2: POSTORDER(TREE->LEFT)
Step-3: PRINT(TREE->ROOT)
Step-4: POSTORDER(TREE->RIGHT)
PRINT(TREE->ROOT)
END OF LOOP
Step-5: EXIT



Therefore, the Preorder traversal of the above tree will be :
7, 1, 0, 3, 2, 5, 4, 6, 9, 8, 10
Therefore, the Postorder traversal of the above tree will be :
0, 2, 4, 6, 5, 3, 1, 8, 10, 9, 7
Therefore, the Inorder traversal of the above tree will be :
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Binary Search Tree

For a binary tree to be a binary search tree, the data of all the nodes in the left sub-tree of the root node should be less than the data of the root. The data of all the nodes in the right subtree of the root node should be greater than equal to the data of the root.



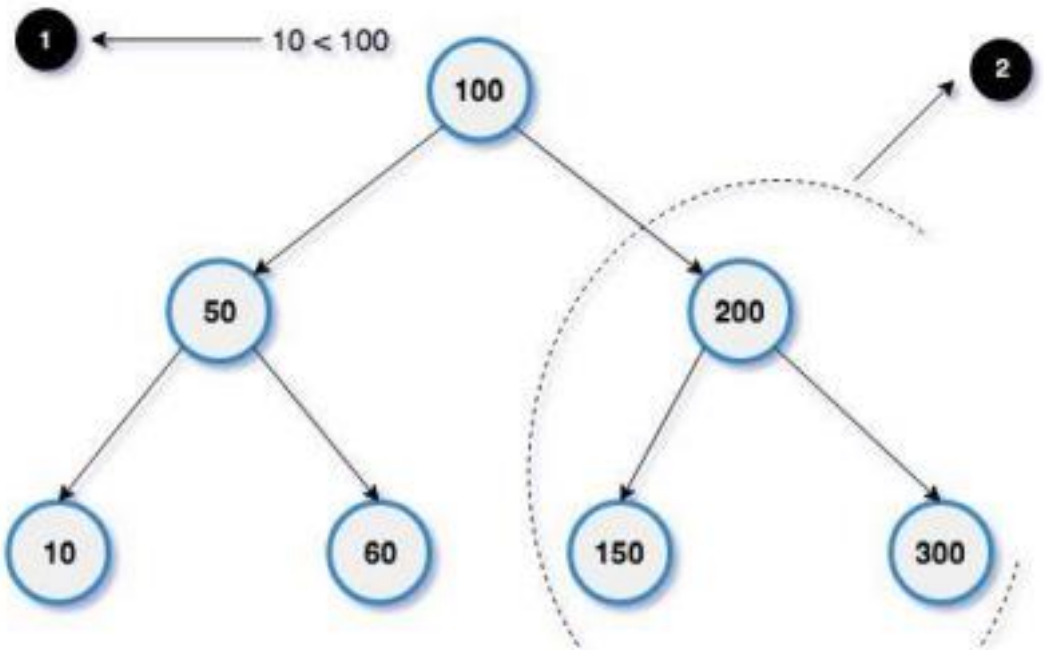
Consider the root node 20. All elements to the left of subtree(10, 5) are less than 20 and all elements to the right of subtree(25, 30, 35) are greater than 20.
Searching Operation

Whenever an element is to be searched,

1. start searching from the root node.
2. Then if the data is less than the key value, search for the element in the left subtree.
3. Otherwise, search for the element in the right subtree.
4. Follow the same algorithm for each node.

1. If we need to search element 10, we can do it effectively like below ,Check if the root node has the value 10. root node value = 100. 10 < 100.

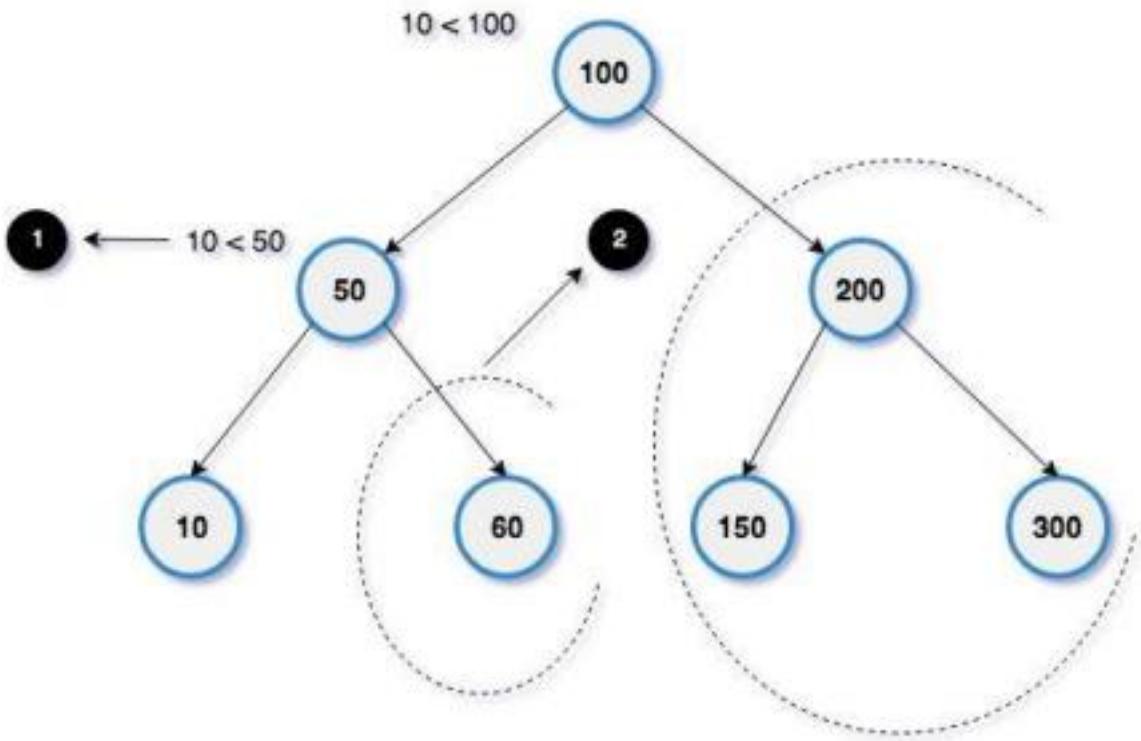
Hence 10 should be present in the left subtree (right subtree must have values > 100). So we can skip entire right subtree from the searching



2.

Check if the left subtree's root has the value 10. Next left root value = 50. again 10 < 50.

So we can skip the right subtree from the searching as the key value less than the root.



6.3 Explain insertion & deletion in a binary search trees

Inserting a node in a binary search tree

Algorithm

Step-1: Create a new BST node and assign values to it.

Step-2: Insert(node, key)

Step-3: Check whether root == NULL, Then

IF(root == NULL)

Step-4: root=newNode

Step-5: ELSEIF Check root=>data > key,THEN

Step-6: Again Check Whether root->left==NULL,THEN

Step-7: root->left=newNode

Call the insert function with root->left and assign the return value in root-> left

Step-8: ELSE Check root=>data < key,THEN

Step-9: Again Check Whether root->right==NULL,THEN

Step-10: root->right=newNode

Call the insert function with root->right and assign the return value in root-> right

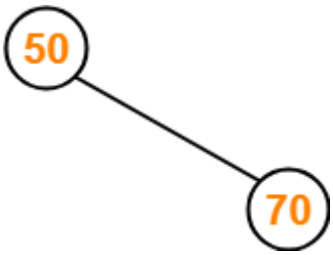
Construct a Binary Search Tree (BST) for the following sequence of numbers-

50, 70, 60, 20, 90, 10, 40, 100

Insert 50-

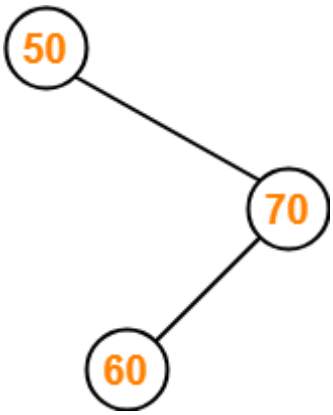
Insert 70-

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



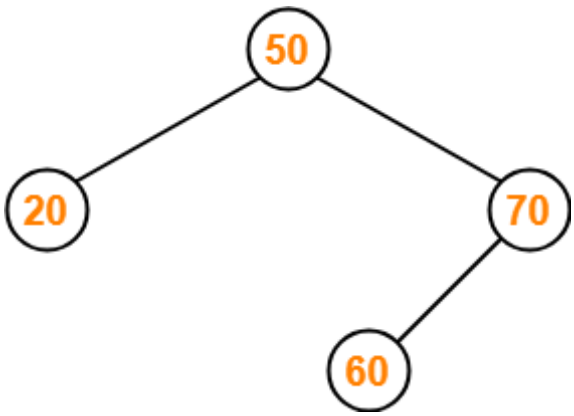
Insert 60-

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



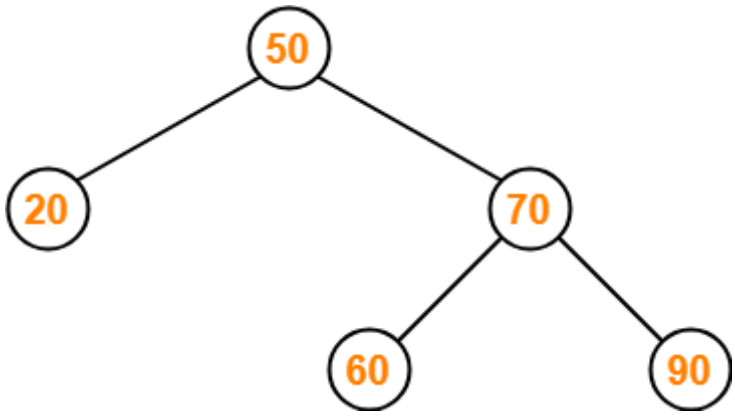
Insert 20-

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



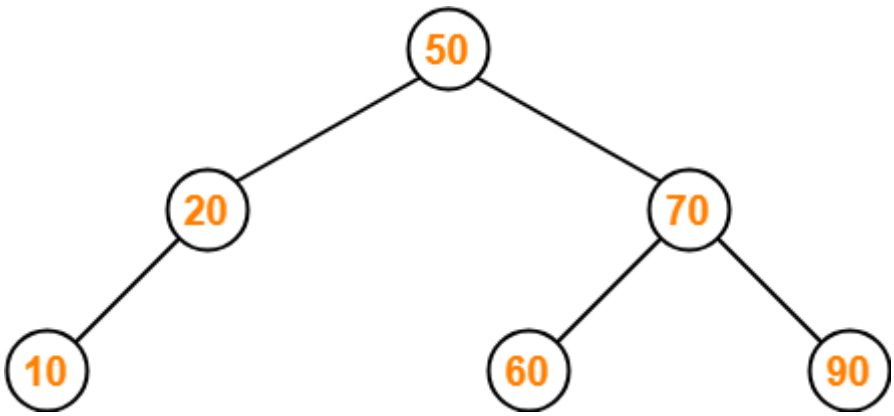
Insert 90-

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



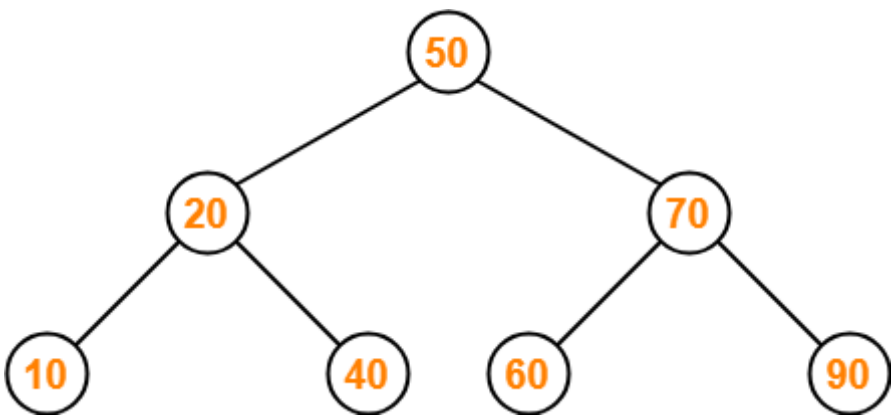
Insert 10-

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



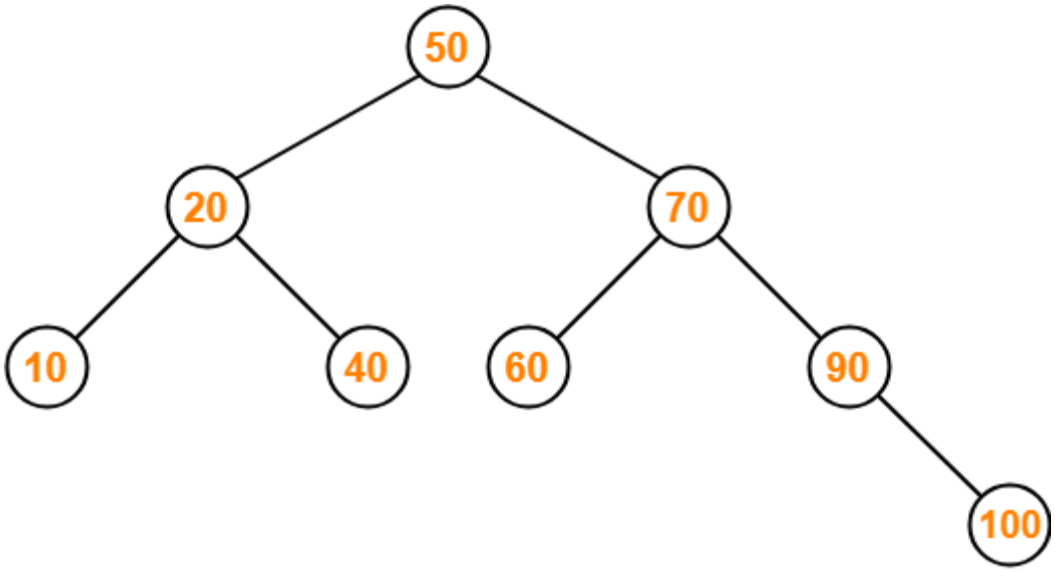
Insert 40-

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



Insert 100-

- As 100 > 50, so insert 100 to the right of 50.
- As 100 > 70, so insert 100 to the right of 70.
- As 100 > 90, so insert 100 to the right of 90.



Binary Search Tree

Deletion in Binary Search Tree

To delete the given node from the binary search tree(BST)

CASE 1 (Leaf Node)

If the node is leaf (both left and right will be NULL), remove the node directly and free its memory.

Example

CASE 2 (Node with Right Child)

If the node has only right child (left will be NULL), make the node points to the right node and free the node.

Example

CASE 3(Node with Left Child)

If the node has only left child (right will be NULL), make the node points to the left node and free the node.

Example

CASE 4 (Node has both left and right child)

If the node has both left and right child,

1.find the smallest node in the right subtree. say min

2.make node->data = min

3.Again delete the min node.

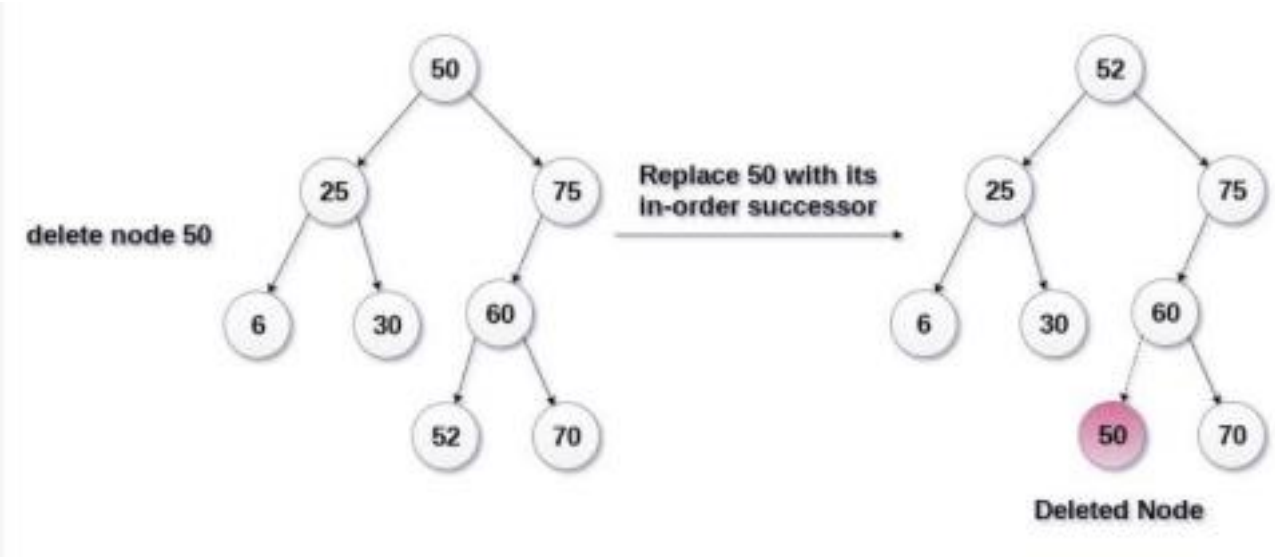
OR

If the node has both left and right child,

1.find the Greatest node in the Left subtree. say max

2.make node->data = max

3.Again delete the max node.



Short Questions

Q1. What is tree?

• A tree is a non-linear data structure that consists of a root node and potentially many levels of additional nodes that form a hierarchy. .

• A tree can be empty with no nodes called the null or empty tree or a tree is a structure consisting of one node called the root and one or more subtrees.

Q2. What is Binary tree? [2015(w),2017(w)]

A binary tree is a hierarchical data structure in which each node has at most two children generally referred as left child and right child.

Q3. What is binary tree traversal? [2015(w)]

1. Inorder
2. postorder
3. preorder

Q4. What is binary search tree? [2014(w)]

For a binary tree to be a binary search tree, the data of all the nodes in the left sub-tree of the root node should be less than the data of the root. The data of all the nodes in the right subtree of the root node should be greater than equal to the data of the root.

Long Questions

Q1. What is BST? Construct a BST with 42,56,23,35,46,66,79,11 ? [2017(w)]

Q2.What do you mean by tree traversal? Diagram to explain different traversal?[2017(w)][2020(w)]

Q3.What What do you mean by BST? How it is different from simple binary tree? Explain how insertion and deletion takes place in a BST with suitable example?

Q3.Explain the procedure of Binary Search?[2019(w)]

Chapter 7
GRAPH

Articles to be covered

7.1 Explain graph terminology & its representation,

7.2 Explain Adjacency Matrix, Path Matrix

7.1 Explain graph terminology & its representation

A Graph is a collection of Vertices(V) and Edges(E).

Vertices also called as nodes. V is a finite set of nodes and it is a nonempty set. Edge refers to the link between two vertices or nodes. So, E is a set of pairs of vertices. In general,
G = (V,E).

where,

G - Graph

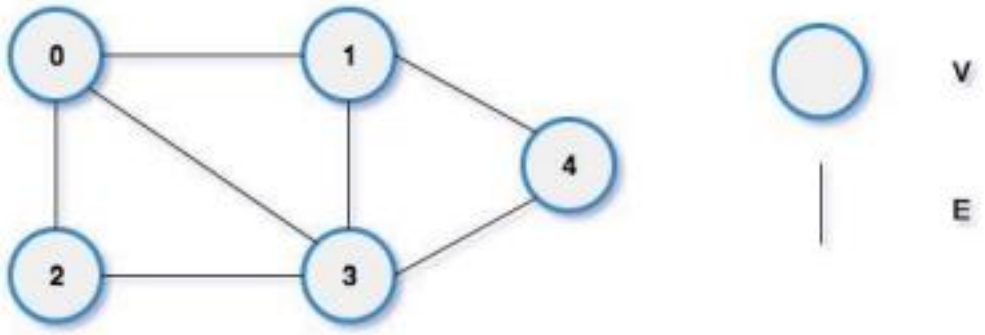
V - a set of nodes. Nonempty set.

E - a set of edges. It can be an empty set.

Undirected Graph

- In Undirected Graph have unordered pair of edges. means edge E1 (x,y) and E2 (y,x) represent the same edge.
- Edge can be traversed from any direction.

Example



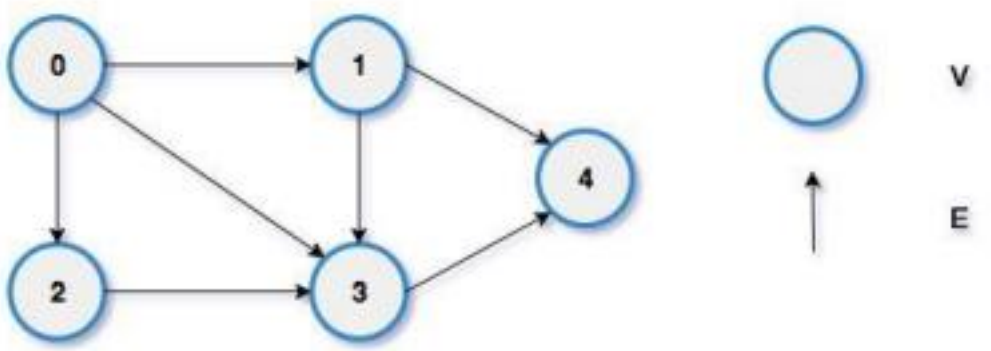
V = {0, 1, 2, 3, 4}

E = {{0,1}, {0,2}, {0,3}, {1,0},{1,3},{1,4}, {2,0},{2,3}, {3,0},{3,1},{3,2},{3,4}, {4,1},{4,3}}

Directed Graph(Digraph)

- In the Directed Graph, each edge(E) will be associated with directions. So, directed Graph have the ordered pair of edges.
- Means edge E1 (x,y) and E2 (y,x) are two different edges.
- Edge can only be traversed from the specified direction.
- It is also called the digraph(Directed graph).

Example



V = {0, 1, 2, 3, 4}

E = {<0,1>, <0,2>, <0,3>, <1,3>,<1,4>,<2,3>,<3,4>}

Terminologies used in graph

A**. A graph G** = (V, E) where

1. V = a set of vertices

2. E = a set of edges

B**. Edges:**

o Each edge is defined by a pair of vertices

o An edge connects the vertices that define it

o In some cases, the vertices can be the same

C. **Vertices:**

o Vertices also called nodes

o Denote vertices with labels

D**. Path**: sequence of vertices in which each pair of successive vertices is connected by an edge

E**. Cycle:** a path that starts and ends on the same vertex

F. **Simple path**: a path that does not cross itself

o That is, no vertex is repeated (except first and last)

o Simple paths cannot contain cycles

G. **Length of a path:** Number of edges in the path

o Sometimes the sum of the weights of the edges

7.2 Explain Adjacency Matrix

Adjacency Matrix

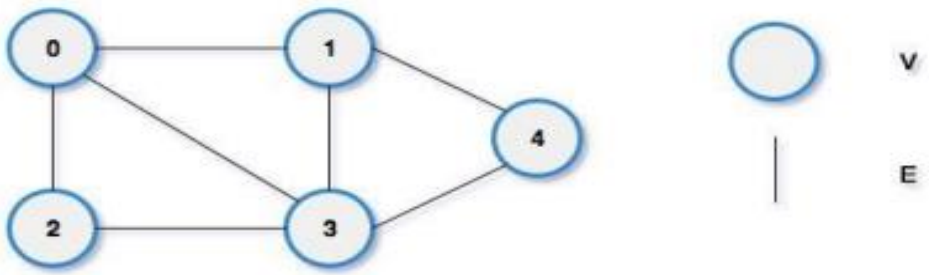
If a graph has n vertices, we use n x n matrix to represent the graph.

Let's assume the n x n matrix as adj[n][n].

- ➢ if there is an edge from vertex i to j, mark adj[i][j] as 1. i.e. adj[i][j] == 1
- ➢ if there is no edge from vertex i to j, mark adj[i][j] as 0. i.e. adj[i][j] == 0

UNDIRECTED GRAPH

Example



V = {0, 1, 2, 3, 4}

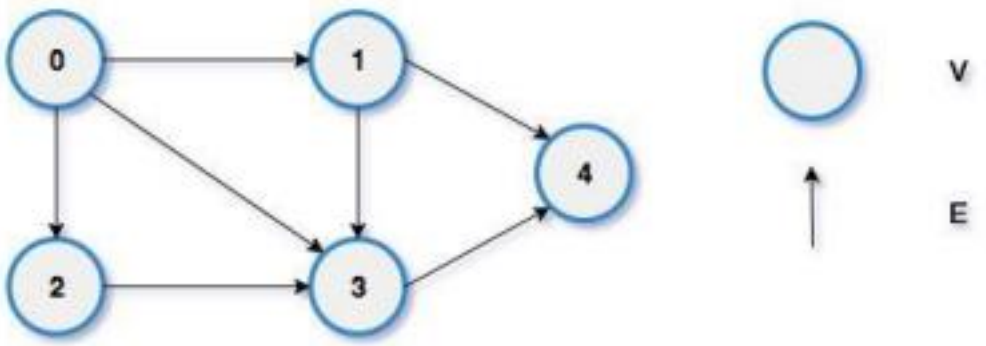
E = {{0,1}, {0,2}, {0,3}, {1,0},{1,3},{1,4}, {2,0},{2,3}, {3,0},{3,1},{3,2},{3,4}, {4,1},{4,3}}

Adjacency Matrix of Undirected Graph

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

DIRECTED GRAPH

Example



V = {0, 1, 2, 3, 4}

E = {<0,1>, <0,2>, <0,3>, <1,3>, <1,4>, <2,3>, <3,4>}

Adjacency Matrix of Directed Graph

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

Path matrix

Let G be a simple directed graph with n nodes ,

V1,V2,V3,……Vn .The path matrix or reachability matrix of graph G is the n-square matrix P=[P_{ij}] is defined as follows

[P_{ij}] =1 if there is a path from node V_i to V_j

0 otherwise

Short Questions

Q1. What is Graph? [2014(w)],[2015(w)],[2018(w)]

- A Graph is a collection of Vertices(V) and Edges(E).
- Vertices also called as nodes. V is a finite set of nodes and it is a nonempty set. Edge refers to the link between two vertices or nodes.
- So, E is a set of pairs of vertices. In general, G = (V,E). where, G - Graph V - a set of nodes. Nonempty set. E - a set of edges. It can be an empty set.

Q2. What is adjacent matrix? [2014(w)][2019(w)]

If a graph has n vertices, we use n x n matrix to represent the graph.
Let's assume the n x n matrix as adj[n][n].

➢ if there is an edge from vertex i to j, mark adj[i][j] as 1. i.e. adj[i][j] == 1
➢ if there is no edge from vertex i to j, mark adj[i][j] as 0. i.e. adj[i][j] == 0

Q3. Define Path matrix?[2020(w)]

Let G be a simple directed graph with n nodes ,

V1,V2,V3,……Vn .The path matrix or reachability matrix of graph G is the n-square matrix P=[P_{ij}] is defined as follows

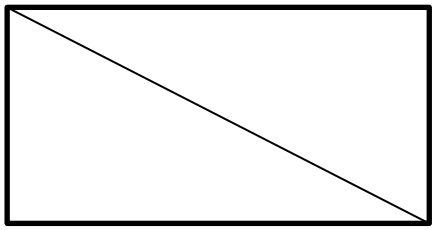
[P_{ij}] =1 if there is a path from node V_i to V_j
0 otherwise

Q4. Define degree of node in a graph?[2016(w)]

- In graph theory, the degree of a vertex of a graph is the no of edges incident to the vertex, with loops counted twice.
- The degree of a vertex (V) is denoted by deg(G).

Long Questions

Q1. Define graph? Draw a graph and write adjacent matrix ?[2020(w)]
Q2. Find the incident matrix of graph?[2018(w)]



Chapter 8
SORTING SEARCHING & MERGING

Articles to be covered

8.1 Discuss Algorithms for Bubble sort, Quick sort,
8.2 Merging
8.3 Linear searching, Binary searching.

8.1 Discuss Algorithms for Bubble sort, Quick sort

SORTING:

Sorting refers to the operation of arranging data in some given order, such as increasing or decreasing with numerical data or alphabetically with character data.

BUBBLE SORT:

- ➢ The bubble sort has no reading characteristics.
- ➢ It is very slow, no matter what data it is sorting
- ➢ As the largest element is bubble of sinks up to its final position. It is known as bubble sort.
- ➢

Algorithm:

BUBBLE (DATA,N)
Here DATA is an array with N element.
This algorithm sorts the element in DATA.

Step 1: [Loop]
Repeat step 2 and step 3 for K=1 to N-1
Step 2: [Initialize pass pointer PTR]
Set[PTR]=1
Step 3: [Execute pass]
Repeat while PTR <=N-K
a. If DATA [PTR] > DATA [PTR+1]
Then interchange DATA [PTR] & DATA [PTR+1]
[End of if structure]

b. Set PTR =PTR+1
[End of Step 1 Loop]
Step 4: Exit

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)

		0	1	2	3	4	5	6	7
i = 0	j								
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i=1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i=2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i=3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i=4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i=5	0	1	2	3	4				
	1	1	2	3					
i=6	0	1	2	3					
		1	2						

Quick sort

It is an algorithm of the divide and conquer type. That is the problem of sorting a set is reduced to the problem of sorting two smaller sets.

Algorithm:-

Step 1 – Choose the highest index value has pivot

Step 2 – Take two variables to point LOW and HIGH of

the list excluding pivot Step 3 – initialize Index of

smaller element as “i” & i value set to (LOW–1) Step 4 –

Traverse element from j = LOW to HIGH–1

Step 5 – while value at j position is less than or equal to pivot then increment “i” & swap value of i &j

Step 6 – while value at i position is greater than pivot then no change.

Step 7:- we place pivot at correct position by swapping arr[i+1] and arr[high]

Example

arr[] = {10, 80, 30, 90, 40, 50, 70}

Indexes: 0 1 2 3 4 5 6

Step 1 &2:- low = 0, high = 6, pivot = arr[h] = 70

Step 3:- Initialize index of smaller element, i = -1

Step 4:-LOOP [Traverse elements from j = low to high-1]

j = 0 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j]) i = 0

arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j are same// j = 1 :

Since arr[j] > pivot, do nothing // No change in i and arr[] j = 2 : Since arr[j]

<= pivot, do i++ and swap(arr[i], arr[j])

i = 1

arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30

j = 3 : Since arr[j] > pivot, do nothing // No change in i and arr[] j = 4 :

Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])

i = 2

arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped

j = 5 : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j]
i = 3

arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped

We come out of loop because j is now equal to high-1.,

Finally we place pivot at correct position by swapping arr[i+1] and arr[high] (or

pivot) arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped

Now 70 is at its correct place. All elements smaller than 70 are before it and all elements greater than 70 are after it.

8.2 MERGING

Merge sort is one of the most efficient sorting techniques and it's based on the "divide and conquer" paradigm.

Algorithm

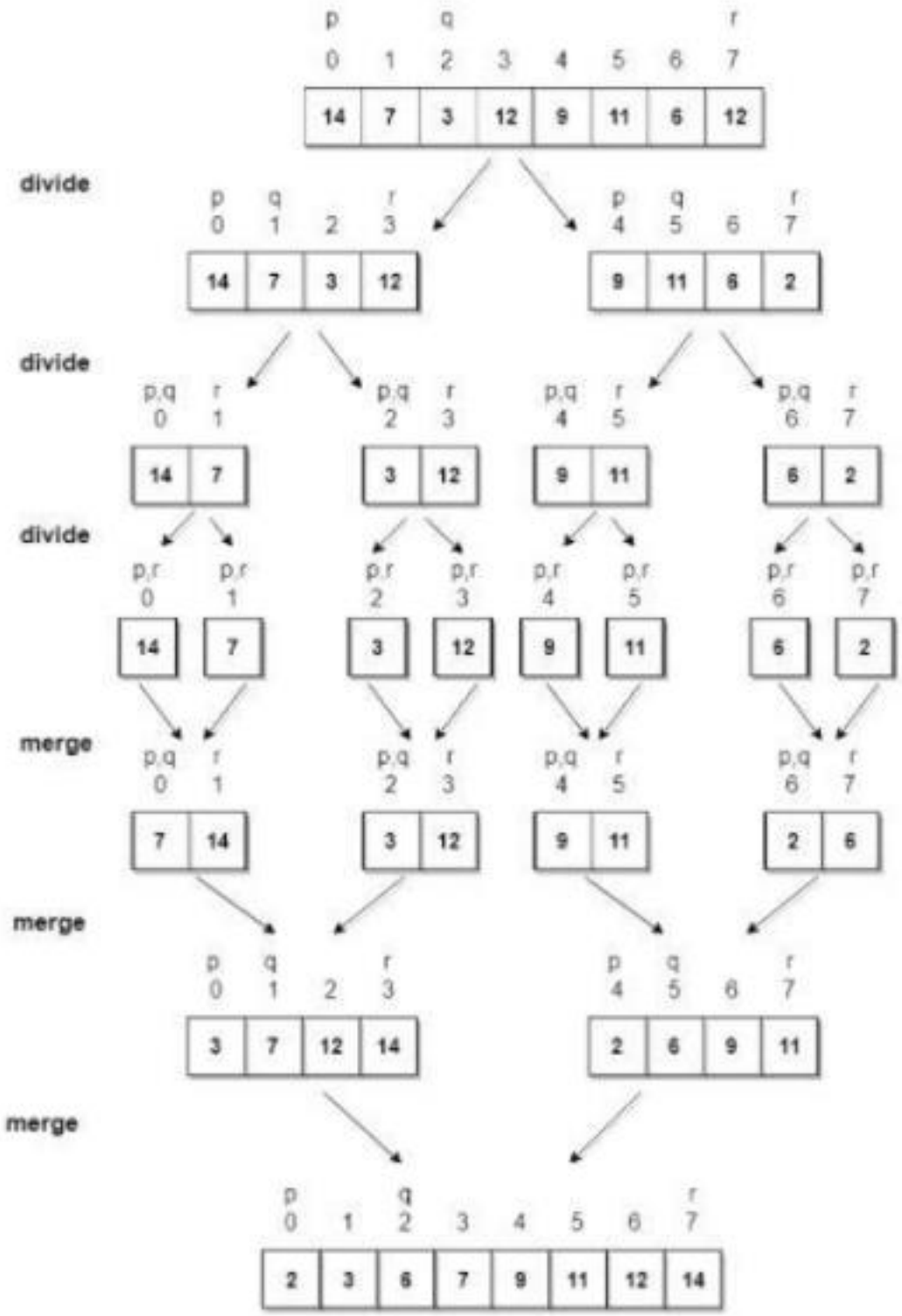
Step 1:- Declare left and right var which will mark the extreme indices of the array

Step2:- Left will be assigned to 0 and right will be assigned to n-1

Step 3:- Find mid = (left+right)/2

Step 4:-Call mergeSort on (left,mid) and (mid+1,rear)

Step 5:- Above will continue till left<right,Then we will call merge on the 2 subproblems
Example



8.3 Linear searching, Binary searching

SEARCHING:

Searching refers to finding the location i.e LOC of ITEM in an array. The search is said to be successful if ITEM appears the array & unsuccessful otherwise we have two types of searching techniques.

1. Linear Search

2. Binary Search

LINEAR SEARCH:

Suppose DATA is a linear array with n elements. No other information about DATA, the most intuitive way to search for a given ITEM in DATA is to compare ITEM with each element of DATA one by one. First we have to test whether DATA [1]=ITEM, nad then we test whether DATA[2] =ITEM , and so on. This method which traverses DATA sequentially to locate ITEM, is called linear search or sequential search.

Algorithm

LINEAR (DATA, N, ITEM, LOC)

Step 1: [Insert ITEM at the end of data]

Set DATA [N+1] = ITEM

Step 2: [Initialize counter]

Set LOC=1

Step 3: [Search for ITEM]

Repeat while DATA [LOC]!= ITEM

Step 4: [Successful]

If LOC=N+1

Then Set LOC = 0

Step 5: Exit

BINARY SEARCH

Suppose DATA is an array which is sorted in increasing numerical order or equivalently, alphabetically. Then there is an extremely efficient searching algorithm, called binary search.

Algorithm

Binary search (DATA, LB, UB, ITEM, LOC)

Step 1: [Initialize the segment variables]

Set BEG := LB, END := UB and MID := INT ((BEG + END)/2)

Step 2: [Loop]

Repeat Step 3 and Step 4 while BEG <= END and DATA [MID] != ITEM

Step 3: [Compare]

If ITEM < DATA [MID]

then set END := MID - 1

Else

Set BEG = MID + 1

Step 4: [Calculate MID]

Set MID := INT ((BEG + END)/2)

Step 5: [Successful search]

If DATA [MID] = ITEM

then set LOC := MID

Else set LOC := NULL

Step 6: Exit

Short Questions

Q1. What is sorting?[2018(w)]

Sorting refers to the operation of arranging data in some given order, such as increasing or decreasing with numerical data or alphabetically with character data.

Q2. What is binary search? [2018(w)]

- Suppose DATA is a linear array with n elements. No other information about DATA, the most intuitive way to search for a given ITEM in DATA is to compare ITEM with each element of DATA one by one.
- This method which traverses DATA sequentially to locate ITEM, is called linear search or sequential search.

Q3. What is linear search?

Suppose DATA is an array which is sorted in increasing numerical order or equivalently, alphabetically. Then there is an extremely efficient searching algorithm, called binary search.

Long Questions

Q1. Write an algorithm for bubble sort?
[2014(w),2015(w),2017(w),2018(w),2019(w),2020(w)]

Q2. Write an algorithm for linear search?[2016(w),2017(w)]

Q3. Write an algorithm for binary search?[2016(w)]

CHAPTER 9
FILE ORGANIZATION

Articles to be covered

9.1 Discuss Different types of files organization and their access method,

9.2 Introduction to Hashing, Hash function, collision resolution, open addressing.

9.1 INTRODUCTION

Nowadays, most organizations use data collection applications which collect large amounts of data in one form or other. For example, when we seek admission in a college, a lot of data such as our name, address, phone number ,the course in which we want to seek admission, aggregate of marks obtained in the last examination and so on, are collected. Similarly, to open a bank account, we need to provide a lot of input. All these data were traditionally stored on paper documents, but handling these documents had always been a chaotic and difficult task. It has become a necessary to store the data in computers in the form of files.

FILE ORGANIZATION

- We know that a file is a collection of related records. The main issue in file management is the way in which the records are organized inside the file because it has a significant effect on the system performance.
- Organization of records means the logical arrangement of records in the file and not the physical layout of the file as stored on a storage media.
- Since choosing an appropriate file organization is a design decision, it must be done keeping the priority of achieving good performance with respect to the most likely usage of the file. Therefore, the following considerations should be kept in mind before selecting an appropriate file organization method:
    - ♦ Rapid access to one or more records.
    - ♦ Ease of inserting/updating/deleting one or more records without disrupting the speed of accessing records(s).
    - ♦ Efficient storage of records.
    - ♦ Using redundancy to ensure data integrity.

TECHNIQUES COMMONLY USED FOR FILE ORGANIZATION

1. Sequential Organization

- A sequential organized file stores the record in the order in which they were entered. That is, the first record that was entered is written as the first record in the file, the second

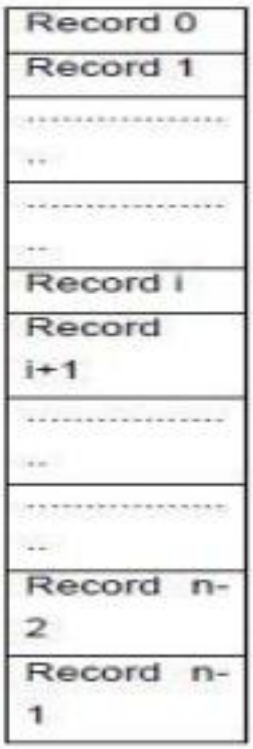
record entered is written as the second record in the file, and so on.

- As a result new records are added only at the end of the file. Sequential files can be read only sequentially, starting with the first record in the file.
- Sequential file organization is the most basic way to organize a large collection of records in a file. The figure below shows n records numbered from 0 to n-1 stored in a sequential file.

- Once we store the records in a file, we cannot make any changes to the records. We cannot even delete the records from a sequential file. In case we need to delete or update one or more records, we have to replace the records by creating a new file.

- In sequential file organization, all the records have the same size and the same field format, and every field has a fixed size. The records are sorted based on the value of one field or a combination of two or more fields. This field is known as the key.

• Each key uniquely identifies a record in a file. Thus, every record has a different value for the key field. Records can be sorted in either ascending or descending order. • Sequential files are generally used to generate reports or to perform sequential reading of large amount of data which some programs need to do such as payroll processing of all the employees of an organization. Sequential files can be easily stored on both disks.



Features	Advantages	Disadvantages
• Records are written in the order in which they are entered. • Records are read and written sequentially. • Deletion or updation of one or more records calls for replacing the original file with a new file that contains the desired changes. • Records have the same size and the same field format. • Records are stored on a key value. • Generally used for report generation or sequential reading.	• Simple and easy to handle. • No extra overheads involved. • Sequential files can be stored on magnetic disks as well as magnetic tapes. • Well suited for batch oriented applications.	• Records can be read sequentially. If 1th record has to be read, then all the i-1 records must be read. • A new file has to be created and the original file has to be replaced with the new file that contains the desired changes. • Cannot be used for interactive application

2. Relative File Organization

- Relative File Organization provides an effective way to access individual records directly in a relative file organization, records are ordered by their relative key. It means the record number represents the location of the record relative to the beginning of the file.
- The record numbers range from 0 to n-1, where n is the number of records in the file. For example, the record with number 0 is the first record in the file. The records in a relative file are of fixed length.
- Therefore, in relative files, records are organized in ascending relative record number. A relative file can be thought of as a single dimension table stored on a disk, in which the relative record number is the index into the table.
- Relative files can be used for both random as well as sequential access. For sequential access, records are simply read one after another. Relative files provide support for only one key, that is, the relative record number.
- This key must be numeric and must take a value between 0 and the current highest relative record number -1. This means that enough space must be allocated for the file to contain the records with relative record numbers between 0 and the highest record number -1.
- For example, if the highest relative record number is 1,000 then space must be allocated to store 1,000 records in the file.
- The figure below shows a schematic representation of a relative file which has been allocated enough space to store 100 records. Although it has space to accommodate 100 records, not all the locations are occupied. The locations marked as FREE are yet to

store records in them. Therefore, every location in the table either stores a record or is marked as FREE.

Relative record number	Records stored in memory
0	Record 0
1	Record 1
2	FREE
3	FREE
4	Record 4
.....	.....
98	FREE
99	Record 99

• Relative file organization provides random access by directly jumping to the record which has to be accessed. If the records are of fixed length and we know the base address of the file and the length of the record, then any record i can be • accessed using the following formula:

Address of ith record=base_address+(i-1)* record_length

• Note that the base address of the file refers to the starting address of the file. We took i-1 in the formula because record numbers start from 0 rather than 1.

• Consider the base address of a file is 1000 and each record occupies 20 bytes,

then the address of the 5th record can be given as: 1000+ (5-1)*20

=1000+80

=1080

Features	Advantages	Disadvantages
• Provides an effective way to access individual records. • The record number represents the location of the record relative to the beginning of the file. • Records in a relative file are of fixed length. • Relative files can be used for both random as well as sequential access.	• Ease of processing. • If the relative record number of the record that has to be accessed is known, then the record can be accessed instantaneously. • Random access of records makes access to relative files fast. • Allow deletions and updations in the same file. • Provides random as well as sequential access of	• Use of relative files is restricted to disk devices. • Records can be of fixed length only. • For random access of records, the relative record number must be known in advance.
• Every location in the table either stores a record or is marked as FREE.	records with low overhead. • New records can be easily added in the free locations based on the relative record number of the record to be inserted. • Well suited for interactive applications.	

3.Indexed Sequential File Organization

- Indexed sequential file organization stores data for fast retrieval. The records in an indexed sequential file are of fixed length and every record is uniquely identified by a key field. We maintain a table known as the index table which stores the record number and the address of all the records.
- That is for every file, we have an index table. This type of file organization is called as indexed sequential file organization because physically the records may be stored anywhere, but the index table stores the address of those records.
- The ith entry in the index table points to the ith record of the file. Initially, when the file is created, each entry in the index table contains NULL. When the ith record of the file is written, free space is obtained from the free space manager and its address is stored inthe ith location of the index table.
- Now, if one has to read the 4th record, then there is no need to access the first three records. Address of the 4th record can be obtained from the index table and the record can be straightaway read from the specified address (742, in our example).

Conceptually, the index sequential file organization can be visualizedas shown in figure.

- An indexed sequential file uses the concept of both sequential file uses the concept of both sequential as well as relative files. While the index table is read sequentially to find the address of the desired record, a direct access is made to the address of the specified record in order to access it randomly.
- Indexed sequential files perform well in situations where sequential access as well as random access is made to the data. Indexed sequential files can be stored only on devices that support random access, for example, magnetic disks.
- For example, take an example of a college where the details of students are stored in an indexed sequential file. This file can be accessed in two ways:

1. Sequentially-to print the aggregate marks obtained by each student in a particular course or
2. Randomly-to modify the name of a particular student.

Record number	Address of the Record	
1	765	Record
2	27	Record
3	876	Record
4	742	Record
5	NULL	
6	NULL	
7	NULL	
8	NULL	
9	NULL	

Features	Advantages	Disadvantages
• Provides fast data retrieval. • Records are of fixed length. • Index table stores the address of the records in the file. • The ith entry in the index table points to the ith record of the file. • While the index table is read sequentially to find the address of the desired record, a direct access is made to the address of the specified record in order to access it randomly. • Indexed sequential files perform well in situations where sequential access as well as random access is made to the data.	• The key improvement is that the indices are small and can be searched quickly, allowing the database to access only the records it needs. • Supports applications that require both batch and interactive processing. • Records can be accessed sequentially as well as randomly. • Updates the records in the same file.	• Indexed sequential files can be stored only on disks. • Needs extra space and overhead to store indices. • Handling these files is more complicated than handling sequential files. • Supports only fixed length records.

9.2 Introduction to Hashing, hash function, collision resolution, open addressing Hashing

1. Hashing is a well-known technique to search any particular element among several elements.
2. It minimizes the number of comparisons while performing the search. **Hashing Mechanism-**

● In hashing, An array data structure called as Hash table is used to store the data items.

● Based on the hash key value, data items are inserted into the hash table.

Hash Key Value-

● Hash key value is a special value that serves as an index for a data item.

● It indicates where the data item should be be stored in the hash table.

● Hash key value is generated using a hash function.

Hash Function-

● Hash function is a function that maps any big number or string to a small integer value.

● Hash function takes the data item as an input and returns a small integer value as an output.
● The small integer value is called as a hash value.

● Hash value of the data item is then used as an index for storing it into the hash table.

Types of Hash Functions

There are various types of hash functions available such as-
● Mid Square Hash Function
● Division Hash Function
● Folding Hash Function etc

It depends on the user which hash function he wants to use.

Collision in Hashing-

● In hashing, Hash function is used to compute the hash value for a key.
● Hash value is then used as an index to store the key in the hash table.
● Hash function may return the same hash value for two or more keys.
● When the hash value of a key maps to an already occupied bucket of the hash table,it is called as a Collision.

```
Say, the set of keys are:
{123, 124, 135, 1267, 2378, 9087}
and hash table size is 10(0-9 ind

Now,
If our hashing function is
F(x)=digits in x

Then
123->3
124->3
135->3
1267->4
2378->4
9087->4
```

The hash table will look like,

Index	Hashed keys
0	
1	
2	
3	123, 124, 135 (collision)
4	1267, 2378, 9087 (collision)
5	
6	
7	
8	
9	

In the above example, we can see though there are 10 indices only 2 are being used and the collision rate is too high. 123, 124, 135 have collided while 1267, 2378, 9087 have collided.

Collision Resolution Techniques

Collision resolution is finding another location to avoid the collision. The most popular resolution techniques are,

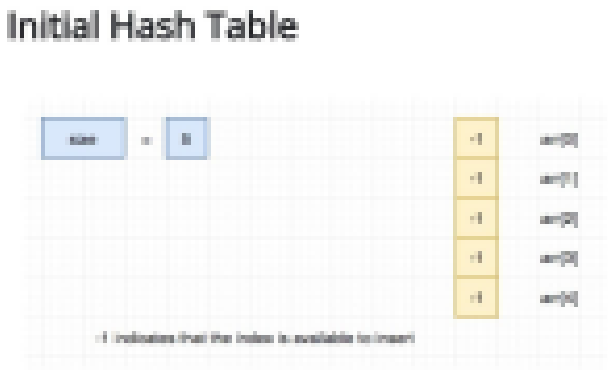
● Separate chaining
● Open addressing

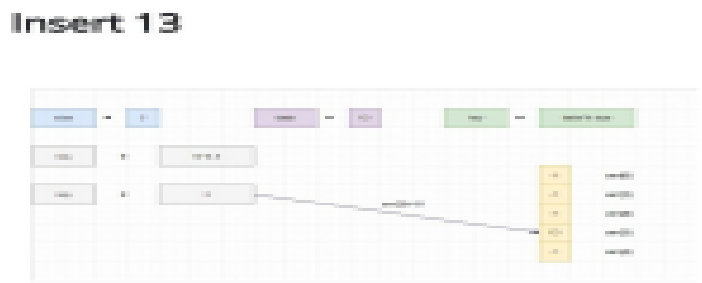
Open addressing can be further divided into,

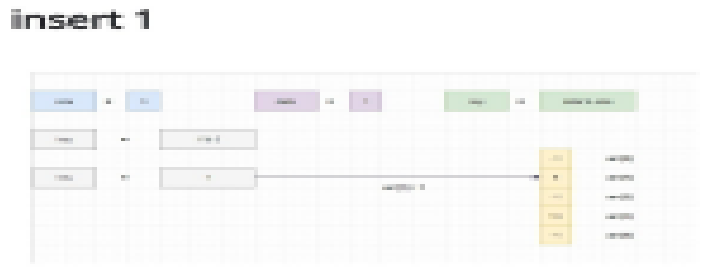
- Linear Probing
- Quadratic Probing
- Double hashing

Open addressing

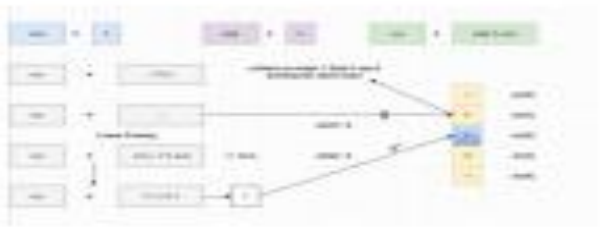
Linear Probing







Insert 6



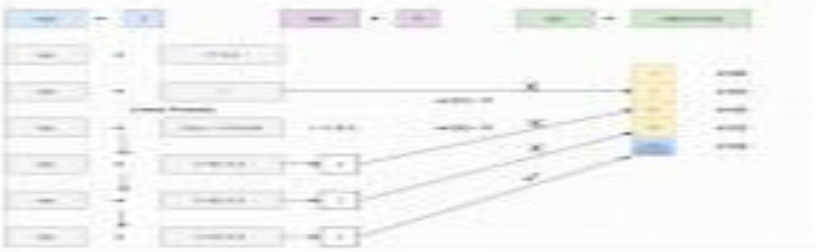
1 % 5 = 1.

6 % 5 = 1.

Both 1 and 6 points the same index under modulo 5.

So that we have placed 6 in arr[2] which is next available index.

Insert 11



1 % 5 = 1.

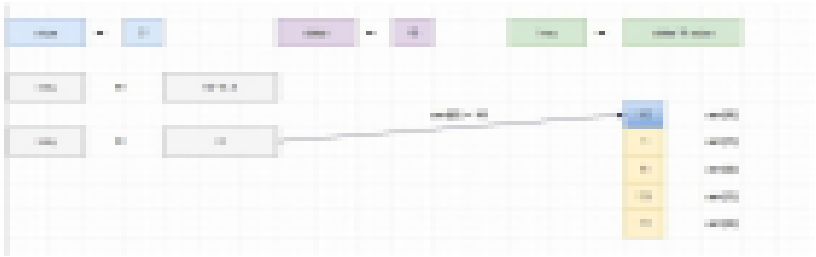
6 % 5 = 1.

11 % 5 = 1.

Both 1, 6 and 11 points the same index under modulo 5.

So that we have placed 11 in arr[4] which is next available index.

Insert 10



2. Quadratic Probing

In quadratic probing,

- When collision occurs, we probe for i2‘th bucket in ith iteration.
- We keep probing until an empty bucket is found.

3. Double Hashing

In double hashing,

- We use another hash function hash2(x) and look for i * hash2(x) bucket in ith iteration.
- It requires more computation time as two hash functions need to be computed.

Short Questions

Q1.What are the techniques for file organization? [2016(w)]

1. Sequential Organization
2. Relative File Organization
3. Indexed Sequential File Organization

Q2. What is hashing? [2014(w)]

- Hashing is a well-known technique to search any particular element among several elements.
- It minimizes the number of comparisons while performing the search.

Q3. What are 3 open addressing technique in hashing?

- Linear Probing
- Quadratic Probing
- Double hashing

Long Questions

Q1.What is Hashing ?Explain different hashing function with example?[2015(w),2017(w)]

Q2. What is collision? Explain various technique to resolve a collision?[2016(w),2018(s),2019(w),2020(w)]