

Model Question Paper-I with effect from 2023-24 (CBCS Scheme)

USN

--	--	--	--	--	--	--	--	--	--

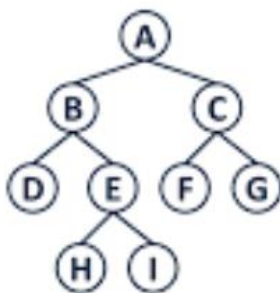
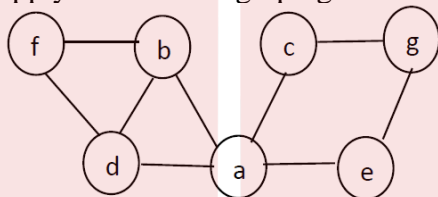
Third Semester B.E. Degree Examination Data Structures and Applications

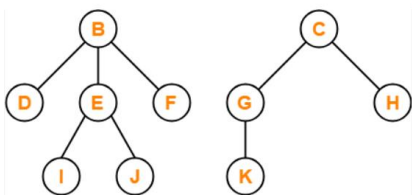
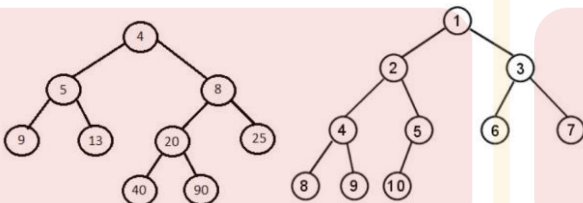
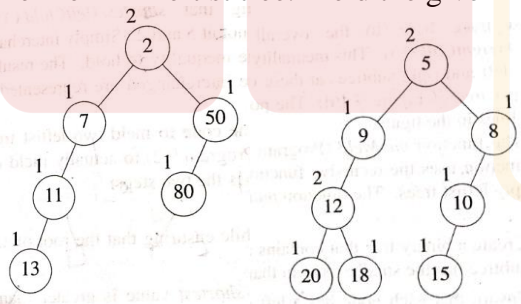
TIME: 03 Hours

Max. Marks: 100

Note: 01. Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE**.

Module -1			*Bloom's Taxonomy Level	Marks
Q.01	a	Define data structures. With a neat diagram, explain the classification of data structures with examples.	L2	5
	b	What do you mean by pattern matching? Outline the Knuth Morris Pratt (KMP) algorithm and illustrate it to find the occurrences of the following pattern. P: ABCDABD S: ABC ABCDAB ABCDABCDABDE	L3	8
	c	Write a program in C to implement push, pop and display operations for stacks using arrays.	L3	7
OR				
Q.02	a	Explain in brief the different functions of dynamic memory allocation.	L2	5
	b	Write functions in C for the following operations without using built-in functions i) Compare two strings. ii) Concatenate two strings. iii) Reverse a string	L3	8
	c	Write a function to evaluate the postfix expression. Illustrate the same for the given postfix expression: ABC-D*+E\$F+ and assume A=6, B=3, C=2, D=5, E=1 and F=7.	L3	7
Module-2				
Q. 03	a	Develop a C program to implement insertion, deletion and display operations on Linear queue.	L3	10
	b	Write a program in C to implement a stack of integers using a singly linked list.	L3	10
OR				
Q.04	a	Write a C program to implement insertion, deletion and display operations on a circular queue.	L3	10
	b	Write the C function to add two polynomials. Show the linked representation of the below two polynomials and their addition using a circular singly linked list P1: $5x^3 + 4x^2 + 7x + 3$ P2: $6x^2 + 5$ Output: add the above two polynomials and represent them using the linked list.	L3	10

Module-3																																			
Q. 05	a	Write recursive C functions for inorder, preorder and postorder traversals of a binary tree. Also, find all the traversals for the given tree. <div></div>	L3	8																															
	b	Write C functions for the following i) Search an element in the singly linked list. ii) Concatenation of two singly linked list	L2	6																															
	c	Define Sparse matrix. For the given sparse matrix, give the linked list representation: $A = \begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$	L3	6																															
OR																																			
Q. 06	a	Write C Functions for the following i) Inserting a node at the beginning of a Doubly linked list Deleting a node at the end of the Doubly linked list	L3	8																															
	b	Define Binary tree. Explain the representation of a binary tree with a suitable example.	L2	6																															
	c	Define the Threaded binary tree. Construct Threaded binary for the following elements: A, B, C, D, E, F, G, H, I	L3	6																															
Module-4																																			
Q. 07	a	Design an algorithm to traverse a graph using Depth First Search (DFS). Apply DFS for the graph given below. <div></div>	L3	8																															
	b	Construct a binary tree from the Post-order and In-order sequence given below In-order: GDHBAEICF Post-order: GHDBIEFCA	L2	6																															
	c	Define selection tree. Construct min winner tree for the runs of a game given below. Each run consists of values of players. Find the first 5 winners. <table data-bbox="260 1776 750 1892"><tr><td>10</td><td>9</td><td>20</td><td>6</td><td>8</td><td>9</td><td>90</td><td>17</td></tr><tr><td>15</td><td>20</td><td>20</td><td>15</td><td>15</td><td>11</td><td>95</td><td>18</td></tr><tr><td>16</td><td>38</td><td>30</td><td>25</td><td>50</td><td>16</td><td>99</td><td>20</td></tr><tr><td colspan="4"></td><td>28</td><td colspan="3"></td></tr></table>	10	9	20	6	8	9	90	17	15	20	20	15	15	11	95	18	16	38	30	25	50	16	99	20					28				L2
10	9	20	6	8	9	90	17																												
15	20	20	15	15	11	95	18																												
16	38	30	25	50	16	99	20																												
				28																															

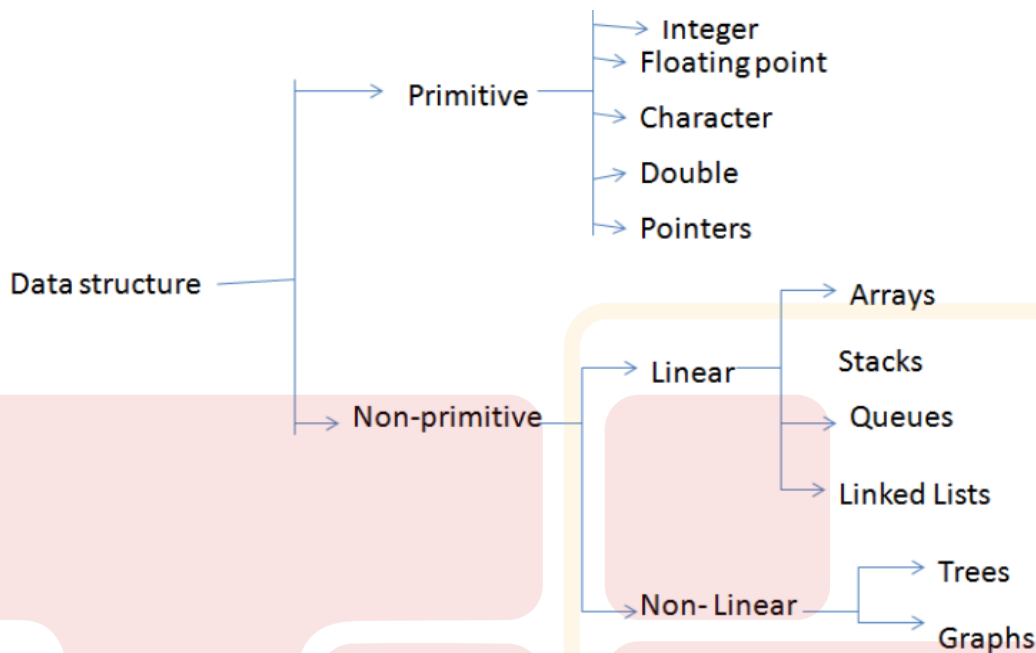
OR				
Q. 08	a	Define Binary Search tree. Construct a binary search tree (BST) for the following elements: 100, 85, 45, 55, 120, 20, 70, 90, 115, 65, 130, 145. Traverse using in-order, pre-order, and post-order traversal techniques. Write recursive C functions for the same.	L3	8
	b	Define Forest. Transform the given forest into a Binary tree and traverse using inorder, preorder and postorder traversal. 	L2	6
	c	Define the Disjoint set. Consider the tree created by the weighted union function on the sequence of unions: union(0,1), union(2,3), union(4,5), union(6,7), union(0,2), union(4,6), and union(0,4). Process the simple find and collapsing find on eight finds and compare which find is efficient.	L2	6
Module-5				
Q. 09	a	What is chained hashing? Discuss its pros and cons. Construct the hash table to insert the keys: 7, 24, 18, 52, 36, 54, 11, 23 in a chained hash table of 9 memory locations. Use $h(k) = k \bmod m$.	L3	10
	b	Define the leftist tree. Give its declaration in C. Check whether the given binary tree is a leftist tree or not. Explain your answer. 	L2	5
	c	What is dynamic hashing? Explain the following techniques with examples: i) Dynamic hashing using directories ii) Directory less dynamic hashing	L2	5
OR				
Q. 10	a	What is a Priority queue? Demonstrate functions in C to implement the Max Priority queue with an example. i) Insert into the Max priority queue ii) Delete into the Max priority queue iii) Display Max priority queue	L3	10
	b	Define min Leftist tree. Meld the given min leftist trees. 	L2	5
	c	Define hashing. Explain different hashing functions with examples. Discuss the properties of a good hash function.	L2	5

1a) Define data structures. With a neat diagram, explain the classification of data structures with examples.

Data Structure: It can be defined as a method of storing and organizing the data items in the computer's memory.

Mainly deal with

- ✓ Storing data in memory
- ✓ Organizing data in memory
- ✓ Fetching and processing data



Primitive data structure The data structures, that are directly operated upon by machine level instructions i.e. the fundamental data types such as > int, > float, > char > double.

A data structure that cannot be manipulated directly by machine instructions are called **non-primitive** data structure . ex: arrays, stacks, queues ,lists Files, trees, graphs.

There are two types of-non primitive data structures. Linear and Non-Linear Data Structures:

In a linear data structure, the data items are arranged in a linear order or sequential. For Example: array, Stack, lists, queue.

In a non-linear data structure, the data items that are not in sequence. For Example: trees and graphs.

An array is a sequence of data item of homogeneous value(same type).

Stacks A stack is a linear data structure in which an element may be inserted or deleted only at one end called the top end of the stack

Queue is a linear data structure in which insertion can take place at only one end called rear end and deletion can take place at other end called front end

A linked list is a data structure which is collection of zero or more nodes with each node consisting of two field's data and link

A tree is a nonlinear data structure and is generally defined as a nonempty finite set of elements, called nodes.

A graph normally a combination of the set of vertices V and set of edges E . $G=(V,E)$

1 b) what do you mean by pattern matching? Outline the Knuth Morris Pratt (KMP) algorithm and illustrate it to find the occurrences of the following pattern.

Pattern: ABCDABD

String: ABC ABCDAB ABCDABCDABDE

It is a fundamental problem in computer science where the goal is to find all occurrences of a given pattern within a larger text or string.

The Knuth-Morris-Pratt (KMP) algorithm is a widely used pattern matching algorithm that efficiently searches for occurrences of a pattern within a text by exploiting the information gathered during the preprocessing phase.

KMP algorithm works in linear time $O(n + m)$, where n is the length of the text and m is the length of the pattern.

Outline of the Knuth-Morris-Pratt algorithm:

Preprocessing: Construct an auxiliary array, called the failure function or failure array, which is used to indicate potential fallback positions in the pattern to avoid redundant comparisons.

Search: Use the failure function to guide the search process, avoiding unnecessary comparisons by efficiently moving through the text.

String: ABC ABCDAB ABCDABCDABDE

Pattern: ABCDABD

1. Align the pattern at the beginning of the string.
2. Compare characters until a mismatch is found or the pattern is exhausted.

String: ABC ABCDAB ABCDABCDABDE

Pattern: ABCDABD

String: ABC ABCDAB ABCDABCDABDE

Pattern: ABCDABD

String: ABC ABCDAB ABCDABCDABDE

Pattern: ABCDABD

String: ABC ABCDAB ABCDABCDABDE

Pattern: ABCDABD

String: ABC ABCDAB ABCDABCDABDE

Pattern: ABCDABD

Example : 2

[illegible]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
String:	a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y
Pattern	i	j	k	l	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	

Algorithm or function (write 2nd lab pgm void string match function)

```
void stringmatch()
{
    while(str[c] !='\0')
    {
        if(str[m] == pat[i])
        {
            i++; m++;
            if(pat[i] == '\0')
            {
                flag = 1;
                for(k=0; rep[k]!='\0'; k++, j++)
                {
                    ans[j] = rep[k];
                }
                c = m;
            }
        }
        else
        {
            ans[j]= str[c];
            j++; c++;
            m=c;
            i=0;
        }
    }
}
```

NOTE : While writing a program kindly maintain { brackets properly }

1c)Write a program in C to implement push, pop and display operations for stacks using arrays.

```
#include<stdio.h>
int stk[10], ch, n, top=-1, item, i;
void push()
{
    if(top>=n-1)
    {
        printf("STACK is over flow\n");
    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d", &item);
        top++;
        stk[top]=item;
        printf(" pushed successfully\n"); } }
```

```
void pop()
{
    if(top == -1)
    {
        printf("Stack is under flow\n");
    }
    else
    {
        printf("The popped elements is %d\n", stk[top]);
        top--;
    }
}
```

```
void display()
{
    if(top == -1)
    {
        printf(" STACK is empty \n");
    }
    else
    {
        printf("The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("%d\n",stk[i]);
    }
}
```

```
void main()
{
    printf("Enter the size of STACK\n");
    scanf("%d", &n);
    for(;;)
    {
        printf("1.PUSH 2.POP 3.DISPLAY \n");
        printf("Enter the Choice:\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: display(); break;
        }
    }
}
```


2 a) Explain in brief the different functions of dynamic memory allocation

Dynamic memory allocation is the process of allocating memory during run time(execution time).

Additional storage can be allocated whenever needed.

1. malloc():

Allocates requested number of bytes and returns a pointer to the first byte of the allocated space.

Syntax:

```
ptr=(datatype *)malloc(sizeof(datatype);
```

where,

- ptr is a pointer variable of type datatype
- datatype can be any of the basic datatype or user define datatype
- Size is number of bytes required.

Example:

```
int *p;  
ptr=(int*)malloc(100*sizeof(int));
```

The calloc Function —

It stands for contiguous allocation. It is used to allocate multiple blocks of memory.

It requires two parameters as number of elements and size of each element.

Syntax: ptr=(datatype *)calloc(n , sizeof(Datatype);

where, ptr is a pointer variable of type datatype

datatype can be any of the basic datatype

n is number of blocks to be allocated size is number of bytes required

```
int *p;  
ptr=(int*)calloc(100,sizeof(int));
```

realloc()

It changes the size of block by deleting or extending the memory at end of the block.

- If memory is not available it gives complete new block.

Syntax:

```
ptr=(datatype *)realloc (ptr , sizeof(datatype);
```

where,

- ptr is a pointer to a block previously allocated memory either using malloc() or calloc()
- Size is new size of the block.

```
int *p;
ptr=(int*)calloc(100,sizeof(int));

ptr=(int*)realloc(ptr,sizeof(int));
```

free()

This function is used to de-allocate(or free) the allocated block of memory which is allocated by using functions malloc(), calloc(), realloc().

Syntax:

```
free(ptr);
```

2b)Write functions in C for the following operations without using built-in functions

i)Compare two strings. ii) Concatenate two strings. iii) Reverse a string

i)Compare two strings

```
int scomp(char s1[], char s2[])
{
    int i, j;
    if(slen(s1) != slen(s2))
    {
        return 0;
    }
    for(i=0; s1[i] != '\0'; i++)
    {
        if(s1[i] != s2[i])
        {
            return 0;
        }
    }
    return 1;
}
```

ii) Concatenate two strings

```
int scat(char s1[], char s2[])
{
    int i, j;
    for(i = slen(s1), j=0; s2[j] != '\0'; i++, j++)
    {
        s1[i] = s2[j];
    }

    s1[i] = '\0';
    return 0;
}
```

iii) Reverse a string

```
void reverse()
{
    char str[100], temp;
    int i, j = 0;
    printf("Enter The String: ");
    gets(str);
    i = 0;
    j = strlen(str) - 1;
    while (i < j)
    {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
    printf("\nReverse a String Is: %s\n", str);
}
```

2c) Write a function to evaluate the postfix expression. Illustrate the same for the given postfix expression: ABC-D*+E\$F+ and assume A=6, B=3, C=2, D=5, E=1 and F=7.

```
#include<stdio.h>
float compute(char symbol, float op1, float op2)
{
    switch (symbol)
    {
        case '+': return op1 + op2;
        case '-': return op1 - op2;
        case '*': return op1 * op2;
        case '/': return op1 / op2;
        case '$':
        case '^': return pow(op1,op2);
    }
}

void main()
{
    float s[20], res, op1, op2;
    int top, i;
    char postfix[20], symbol;
    printf("\nEnter the postfix expression:\n");
    scanf ("%s", postfix);
    top=-1;
    for (i=0; i<strlen(postfix) ;i++)
    {
        symbol = postfix[i];
        if(isdigit(symbol))
            s[++top]=symbol - '0';
        else
        {
            op2 = s[top--];
            op1 = s[top--];
            res = compute(symbol, op1, op2);
            s[++top] = res;
        }
    }
    res = s[top--];
    printf("\nThe result is : %f\n", res);
}
```

ABC-D*+E\$F+

632-5*+1\$7+

Ans : 8

Refer class notes for solving the steps you need to write the steps also

3a Develop a C program to implement insertion, deletion and display operations on Linear queue

```
#include<stdio.h>
int q[10], ch, size, front=-1, rear=-1, item, i;
void enqueue()
{
    if(rear == size - 1)
        printf("Queue Overflow\n");
    else
    {
        if(front== - 1)
        front = 0;
        printf("Inset the element in queue\n: ");
        scanf("%d", &item);
        q[rear++] = item;
    }
}
void dequeue()
{
    if(front== -1 || front>rear)
    {
        printf("Queue Underflow\n");
    }
    else
    {
        printf("Element deleted from queue is : %d\n", q[front]);
        front++;
    }
}
void display()
{
    if(front== -1 || front>rear)
    {
        printf("Queue is empty\n");
    }
    else
    {
        printf("Queue is :\n");
        for(i = front; i <= rear; i++)
            printf("%d\n", q[i]);
    }
}
void main()
{
    printf("Enter the size of STACK\n");
    scanf("%d", &size);
    for(;;)
    {
        printf("1.insert 2.delete 3.DISPLAY \n");
        printf("Enter the Choice:\n");
```

```

scanf("%d",&ch);
switch(ch)
{
case 1: enqueue(); break;
case 2: dequeue();break;
case 3: display(); break;
}
}
}

```

3b Write a program in C to implement a stack of integers using a singly linked list. (dsa lab pgm 7 as it is except data members take only one int data and write as it is code of lab pgm 7)

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
int data ;
struct node* next;
}*tmp,*ptr,*head=NULL;

struct node* createnode()
{
ptr=(struct node*)malloc(sizeof(struct node));
printf("Enter data \n");
scanf("%d",&ptr->data);
ptr->next=NULL;
return ptr;
}

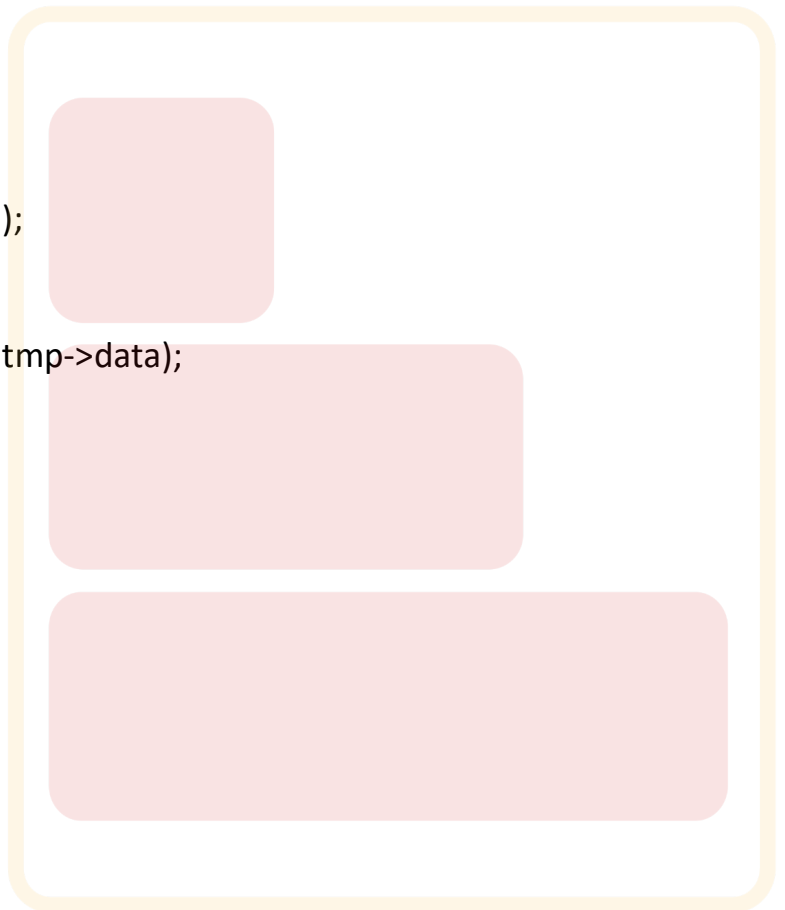
void insertfront()
{
ptr=createnode();
if(head!=NULL)
ptr->next=head;
head=ptr;
}

```

```

void insertend()
{
ptr=createnode();
ptr->next=NULL;
tmp=head;
if (head!=NULL)
{
while(tmp->next!=NULL)
tmp=tmp->next;
tmp->next=ptr;
}
else
head=ptr;
}
void delfront()
{
tmp=head;
if(head==NULL)
printf("THE LINKED LIST IS EMPTY\n");
else
{
printf("THE DELETED NODE is %d\n" tmp->data);
head=tmp->next;
free(tmp);
}
}
void delend()
{
ptr=head;
if (head==NULL)
printf("LINKED LIST IS EMPTY\n");
else
{
while(ptr->next->next!=NULL)
ptr=ptr->next;
tmp=ptr->next;
ptr->next=NULL;
printf("deleted node is %d",tmp->data);
free(tmp);
}
}

```



```
}
```

```
void display()
```

```
{
```

```
if(head==NULL)
```

```
{
```

```
printf("List is empty!!");
```

```
}
```

```
else
```

```
{
```

```
tmp=head;
```

```
int c=0;
```

```
while (tmp!=NULL)
```

```
{
```

```
printf("%d\n tmp->data);
```

```
tmp=tmp->next;
```

```
c++;
```

```
}
```

```
printf("THE NO. OF NODES ARE %d\n",c);
```

```
}
```

```
}
```

```
void main()
```

```
{
```

```
int ch ;
```

```
for(;;)
```

```
{
```

```
printf("Enter your choice\n");
```

```
printf("1-create 2-display and count 3-Insert front\n 4.insert end \n 5.Delete front\n6.deletion  
from end \n");
```

```
scanf("%d",&ch);
```

```
switch(ch)
```

```
{
```

```
case 1:create(); break;
```

```
case 2: display(); break;
```

```
case 3: insertfront(); break;
```

```
case 4: insertend(); break;
```

```
case 5: delfront(); break;
```

```
case 6: delend(); break;
```

```
}
```

```
}
```

```
}
```

4 a Write a C program to implement insertion, deletion and display operations on a circular queue (dsa lab pgm 6th)

```
#include <stdio.h>
int cq[10], front = -1, rear = -1, item, ch, i, size;
```

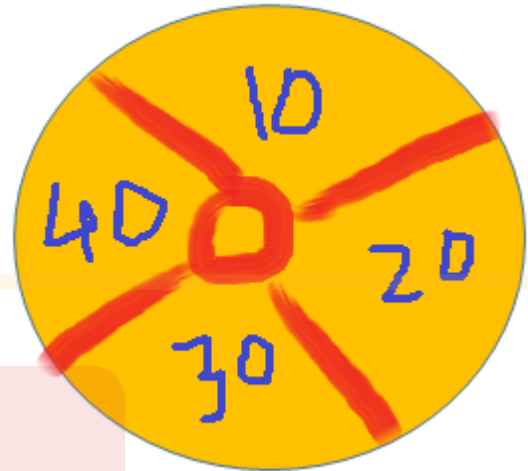
```
void enQueue()
{
if(front == (rear+1)%SIZE)
{
printf("C Q IS OVERFLOW\n");
}
```

```
else
{
printf("enter element\n");
scanf("%d",&item);
if (front == -1)
front = 0;
rear = (rear + 1) % SIZE;
cq[rear] = item;
printf("Inserted\n");
}
```

```
void deQueue()
{
if(front == -1)
{
printf("CIRCULAR QUEUE IS UNDERFLOW\n");
}
else
{
printf("\n Deleted element -> %d \n", cq[front]);
if (front == rear)
front = rear = -1;
else

```

```
front = (front + 1) % SIZE;
}
}
```




```
void display()
{
    if(front == -1)
    {
        printf("CIRCULAR QUEUE IS empty\n");
    }
    else
    {
        printf("Items are \n");
        for (i = front; i != rear; i = (i + 1) % SIZE)
        {
            printf("%d\n", cq[i]);
        }
        printf("%d\n", cq[i]);
    }
}

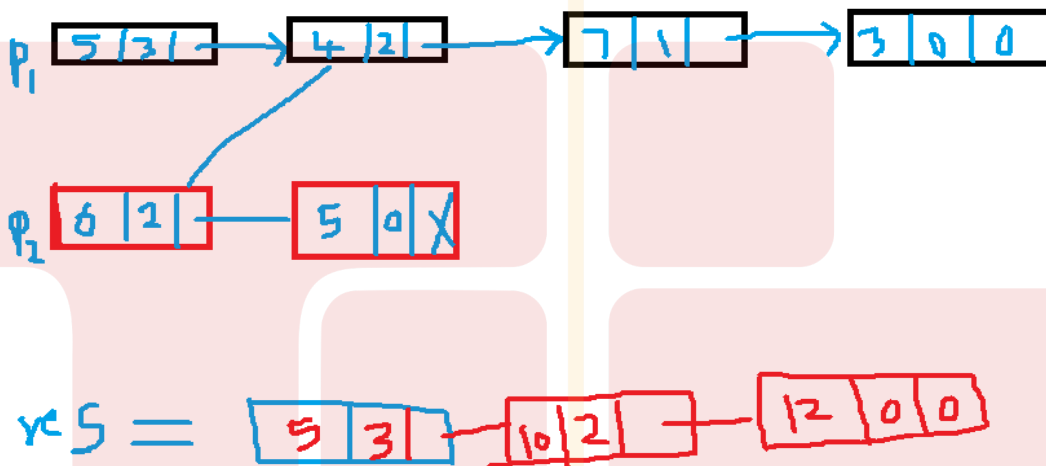
void main()
{
    for(;;)
    {
        printf("enter cq size !!!\n");
        scanf("%d",&size);
        printf("1.Insert 2.delete 3.display 4.Exit\n");
        printf("Enter your choice:\n");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:enQueue();break;
            case 2:deQueue();break;
            case 3:display();break;
        }
    }
}
```

4b Write the C function to add two polynomials. Show the linked representation of the below two polynomials and their addition using a circular singly linked list P1: $5x^3 + 4x^2 + 7x + 3$ P2: $6x^2 + 5$ Output: add the above two polynomials and represent them using the linked list

```
poly_pointer padd(poly_pointer a, poly_pointer b)
{
    poly_pointer c, rear, temp;
    int sum;
    rear = (poly_pointer)malloc(sizeof(poly_node));
    front = rear;
    while (a && b)
    {
        switch (COMPARE(a->expon, b->expon))
        {
            case -1: /* a->expon < b->expon */
                attach( b->coef, b->expon, &rear);
                b = b->link; break;

            case 0: /* a->expon == b->expon */
                sum = a->coef + b->coef;
                if (sum) attach( sum, a->expon, &rear);
                a = a->link; b = b->link; break;

            case 1: /* a->expon > b->expon */
                attach( a->coef, a->expon, &rear);
                a = a->link; break;
        }
    }
}
```



5a Write recursive C functions for inorder, preorder and postorder traversals of a binary tree. Also, find all the traversals for the given tree.

```
void inorder(struct node *root)
{
    if(root != NULL)
    {
        inorder(root->left);
        printf("%d\t", root->data);
        inorder(root->right);
    }
}
```

In order : D B H E I A F C G (APPLY formula and write)

Preorder: A B D E H I C F G

Post order : D H I E B F G C A

```

void preorder(struct node *root)
{
    if(root != NULL)
    {
        printf("%d\t", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

```

```

void postorder(struct node *root)
{
    if(root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d\t", root->data);
    }
}

```

5 b) Write C functions for the following i) Search an element in the singly linked list. ii) Concatenation of two singly linked list

i) Search

```

void search(struct node *head,int key)
{
    struct node *temp = head;
    while(temp != NULL)
    {
        if(temp->data == key)
            printf("key found");
        temp = temp->next;
    }
    printf("key not found");
}

```

ii) Concatenation

```

void Concat(struct Node *first, struct Node *second)
{
    struct Node *p = first;
    while (p->next != NULL)
    {
        p = p->next;
    }
    p->next = second;
    second = NULL;
}

```

5c Define Sparse matrix. For the given sparse matrix, give the linked list representation:

Sparse matrix is a special matrix made of **m rows and n columns**, therefore having total $m \times n$ values with most of its elements are zero.

We can also assume that if $(m * n) / 2$ elements are **zero** then it is a sparse matrix.

Draw the diagram

6a) Write C Functions for the following

- i) **Inserting a node at the beginning of a Doubly linked list**
- ii) **Deleting a node at the end of the Doubly linked list**

Note : refer DSA LAB PGM 8 only functions

```
void insertfront()
{
    ptr=createnode();
    if(start==NULL)
    {
        ptr->left=ptr->right=NULL;
        start=last=ptr;
    }
    else
    {
        ptr->left=NULL;
        ptr->right=start;
        start->left=ptr;
        start=ptr;
    }
}
```

```
void deleteend()
{
    temp=start;
    while(temp->right!=NULL)
    {
        temp=temp->right;
    }
    last=temp->left;
    last->right=NULL;
    temp->left=NULL;
    printf("element is deleted \n");
    free(ptr);
}
```

6b Define Binary tree. Explain the representation of a binary tree with a suitable example.

Definition: A binary tree T is defined as a finite set of nodes such that,

- T is empty or
- T consists of a root and two disjoint binary trees called the left sub tree and the right sub tree.

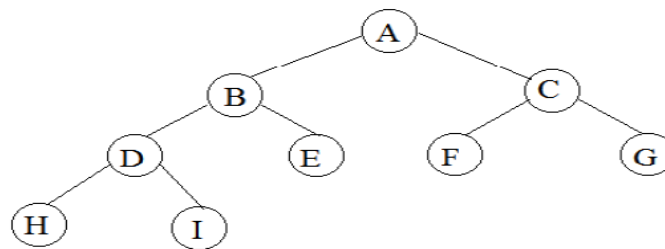


Figure: Binary Tree

Representation of a binary tree:

1. Array representation
2. Linked representation

Array representation:

- A tree can be represented using an array, which is called sequential representation.
- The nodes are numbered from 1 to n , and one dimensional array can be used to store the nodes.
- Position 0 of this array is left empty and the node numbered i is mapped to position i of the array.

Root node $a[0]$
Left Child node $2i+1$
Right Child node $2i+2$

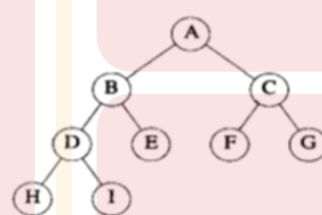


Figure 1(b) Complete binary tree

Root node $\rightarrow a[0]$
Left Child node $\rightarrow 2i+1$
Right Child node $\rightarrow 2i+2$

tree	
[0]	A
[1]	B
[2]	C
[3]	D
[4]	E
[5]	F
[6]	G
[7]	H
[8]	I
[9]	.
.	.
.	.
[16]	

Linked representation: Each node has three fields,

- LeftChild - which contains the address of left subtree
- RightChild - which contains the address of right subtree.
- Data - which contains the actual information

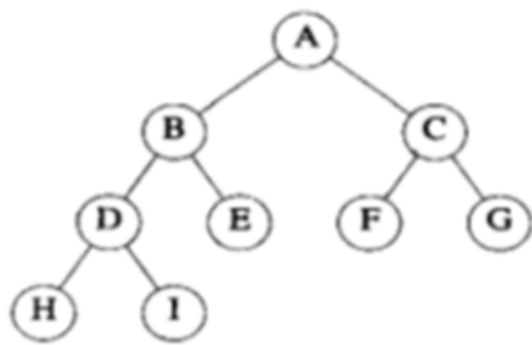
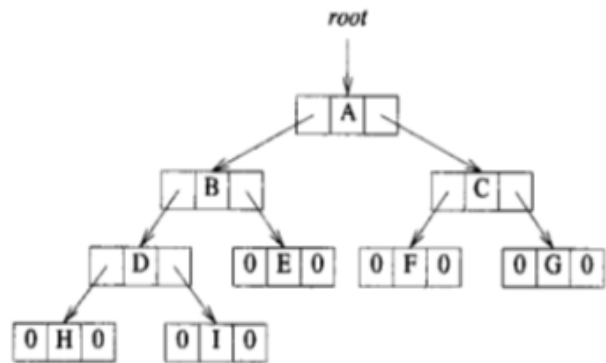


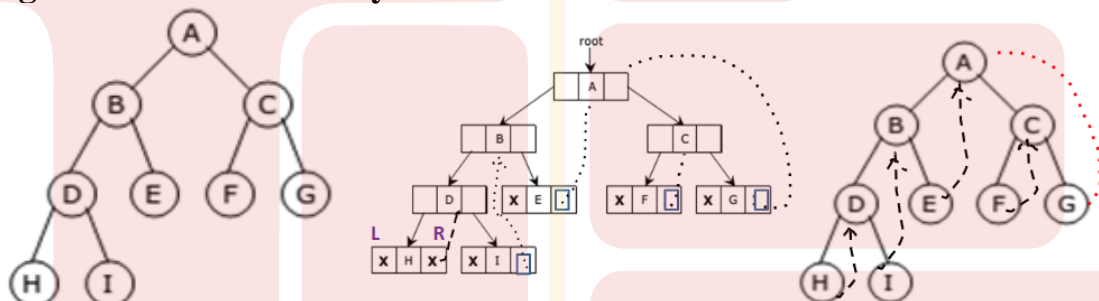
Figure 1(b) Complete binary tree



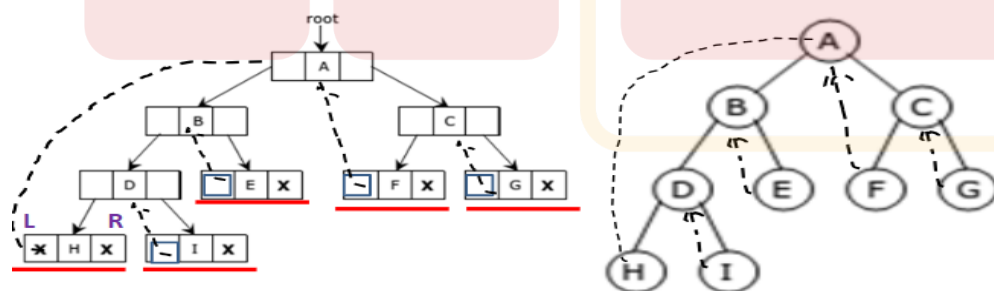
6 c) Define the Threaded binary tree. Construct Threaded binary for the following elements: A, B, C, D, E, F, G, H, I

In a threaded binary tree, each node has an additional pointer called a thread, which points to either its in-order predecessor or successor. This allows us to efficiently traverse the tree without using recursion or a stack.

Right in Threaded binary tree



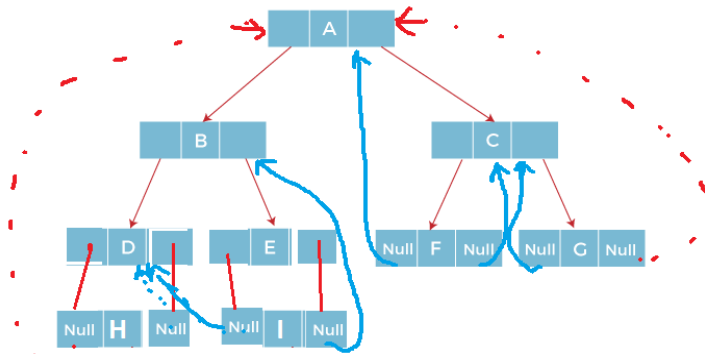
Left in Threaded binary tree



In order traversal

H D I B E A F C G

If there is no in-order predecessor or in-order successor, then it points to root node.



diagram

Refer class notes for this full

7 a Design an algorithm to traverse a graph using Depth First Search (DFS). Apply DFS for the graph given below.

Depth First Search: Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search

DFS Algorithm :

Step 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

Step 2 – If no adjacent vertex is found, pop up a vertex from the stack. (which do not have adjacent vertices.)

Step 3 – Repeat Rule 1 and Rule 2 until the stack is empty

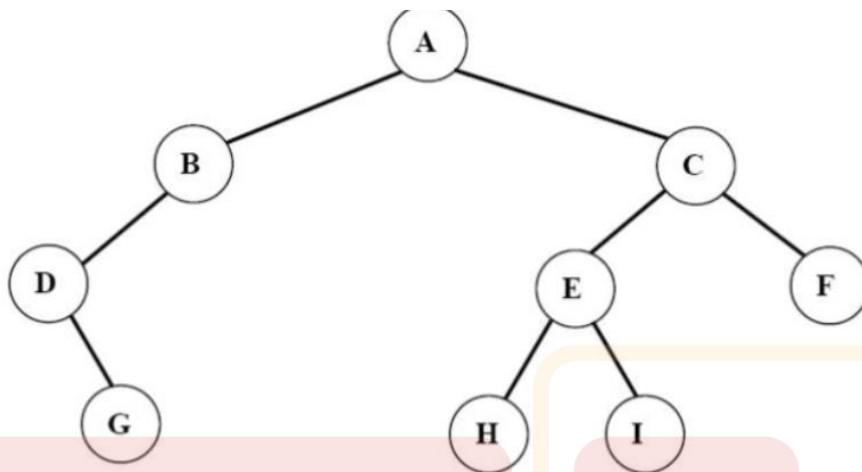
Draw the diagrams

7 b) Construct a binary tree from the Post-order and In-order sequence given below

In-order: GDHBAEICF

Post-order: GHDBIEFCA

Refer class notes module-4 (step by step solution is required to construct this tree)



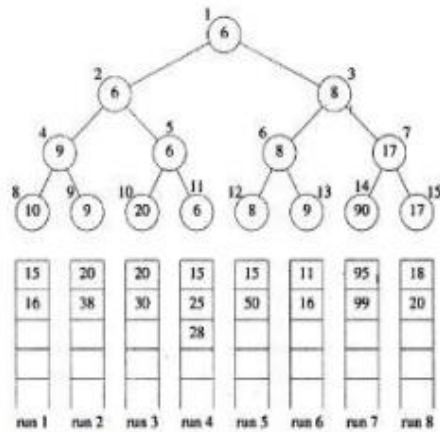
7c Define selection tree. Construct min winner tree for the runs of a game given below. Each run consists of values of players. Find the first 5 winners

SELECTION TREE

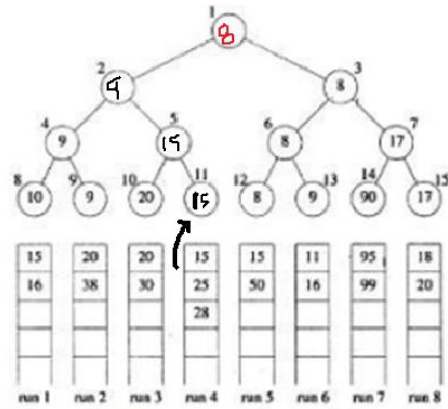
- This is also called as a tournament tree. This is such a tree data structure using which the winner (or loser) of a knock out tournament can be selected.
- There are two types of selection trees namely: winner tree and loser tree.

WINNER TREE

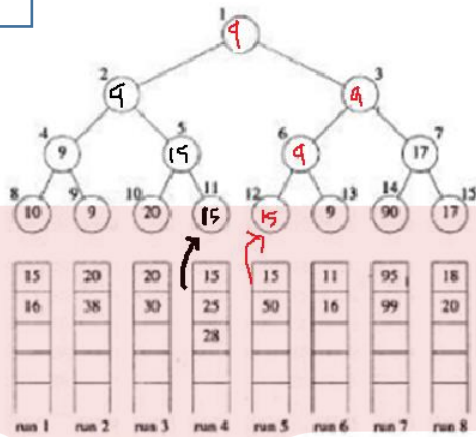
- This is a complete binary tree in which each node represents the smaller of its two children. Thus, the root node represents the smallest node in the tree.



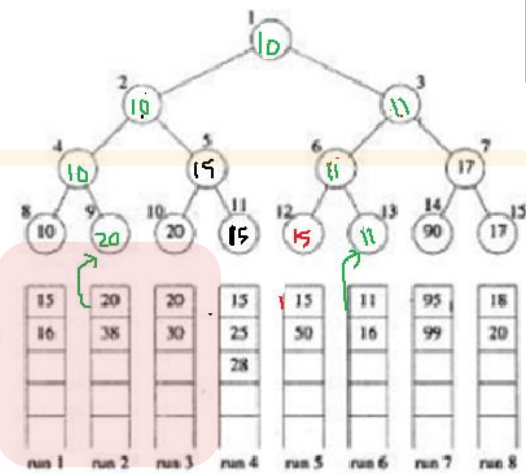
6,8



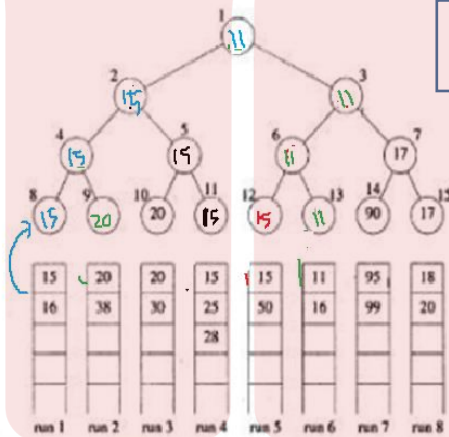
6,8,9



6,8,9,10



6,8,9,10,11



first 5 winners : 6,8,9,10,11

After getting smaller number write down in the output and delete that number from the tree node and enter new number from the given box .

Ex: fig -1 6 we got write in output box as 6

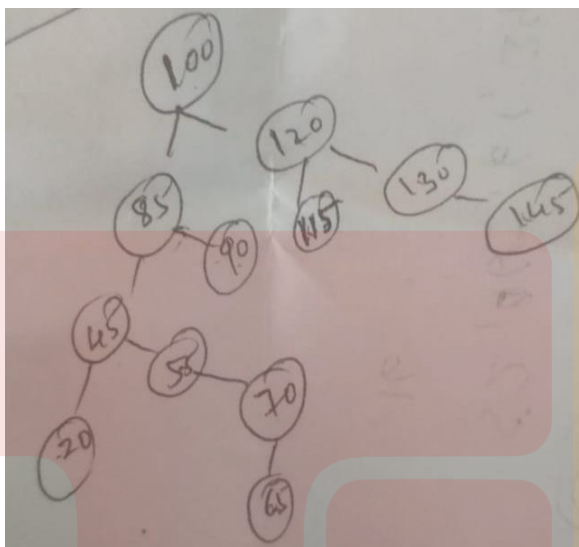
Draw same diagram in node 6 we have to delete the number and insert new number from corresponding the box same.ie 15

8a Define Binary Search tree. Construct a binary search tree (BST) for the following elements: 100, 85, 45, 55, 120, 20, 70, 90, 115, 65, 130, 145. Traverse using in-order, pre-order, and post-order traversal techniques. Write recursive C functions for the same.

Definition of binary search tree:

- Every element has a unique key
- The keys in a nonempty **left subtree** (**right subtree**) are **smaller** (**larger**) than the key in the root of subtree
- The left and right subtrees are also binary search trees

Construction of binary search tree



Recursive C functions

Refer answer from 5a

8b Define Forest. Transform the given forest into a Binary tree and traverse using inorder, preorder and postorder traversal.

FORESTS

- This is a set of $n \geq 0$ disjoint trees (Figure 5.34).

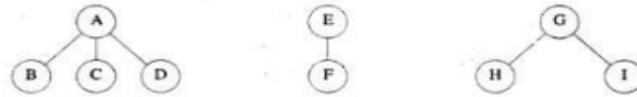
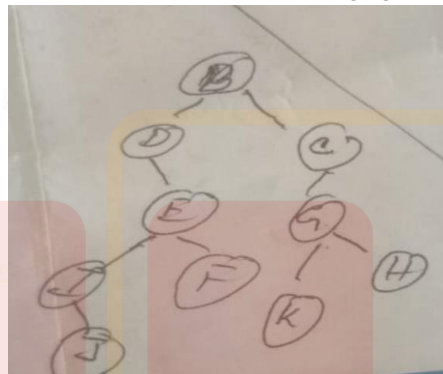


Figure 5.34: Three-tree forest

TRANSFORMING A FOREST INTO A BINARY TREE

- If T_1, T_2, \dots, T_n is a forest of trees, then the binary tree corresponding to this forest, denoted by $B(T_1, T_2, \dots, T_n)$,
 - is empty if $n=0$
 - has root equal to $\text{root}(T_1)$; has left subtree equal to $B(T_{11}, T_{12}, \dots, T_{1m})$
 - has right subtree $B(T_2, \dots, T_n)$ where $T_{11}, T_{12}, \dots, T_{1m}$ are the subtrees of $\text{root}(T_1)$.



FOREST TRAVERSALS

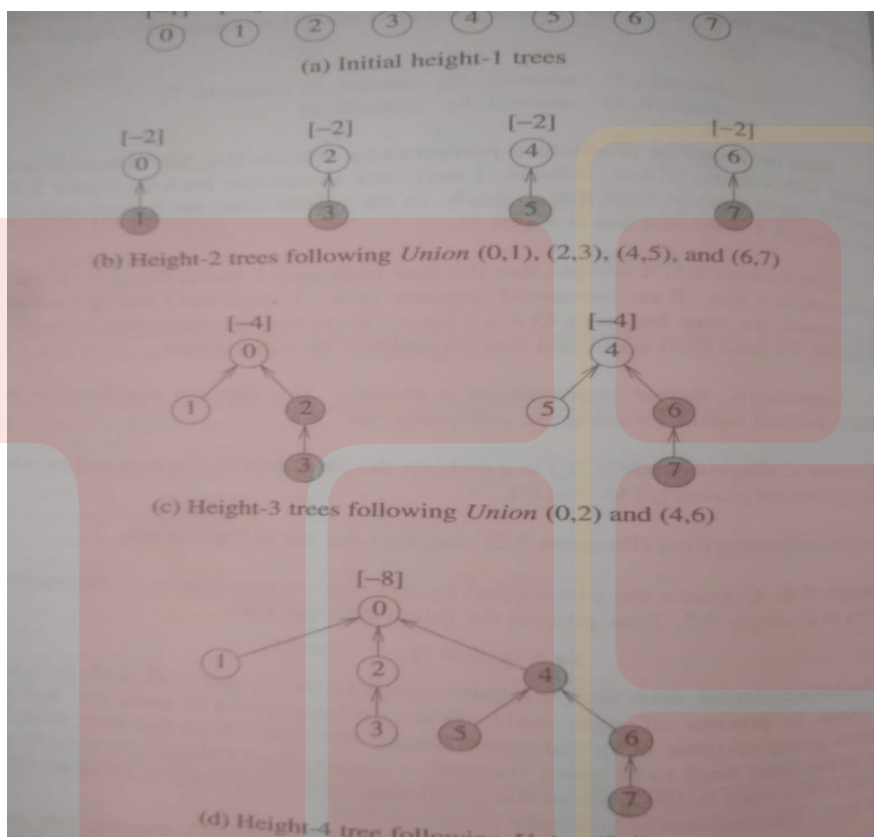
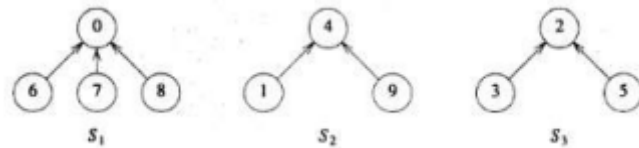
- There are 3 forest traversal techniques namely: preorder, inorder and postorder traversal.
- Preorder traversal of forest F can be recursively defined as follows
 - If F is empty then return.
 - Visit the root of the first tree of F.
 - Traverse the subtrees of the first tree in forest preorder.
 - Traverse the remaining trees of F in forest preorder.
- Inorder traversal of forest F can be recursively defined as follows
 - If F is empty then return.
 - Traverse the subtrees of the first tree in forest inorder.
 - Visit the root of the first tree of F.
 - Traverse the remaining trees of F in forest inorder.
- Postorder traversal of forest F can be recursively defined as follows
 - If F is empty then return.
 - Traverse the subtrees of the first tree in forest postorder.
 - Traverse the remaining trees of F in forest postorder.
 - Visit the root of the first tree of F.

Hint: same for LVR, VLR, LRV

Only if is empty then return
this line you should write

8c Define the Disjoint set. Consider the tree created by the weighted union function on the sequence of unions: union(0,1), union(2,3), union(4,5), union(6,7), union(0,2), union(4,6), and union(0,4). Process the simple find and collapsing find on eight finds and compare which find is efficient.

A set is a collection of elements. Disjoint sets are such sets in which common elements are not present. **Examples:**



Find(i): Find the set containing the element i.

```

Algorithm find(i)
{
  while(p[i] >= 0)
  {
    i = p[i];
  }
  return i;
}

```

Algorithm **collapsing_find(i)**

```
{
    r=i
    while(p[r]>0)
    {
        r=p[r]
    }
    while(i!=r)
    {
        s=p[i]
        p[i]=r
        i=s
    }
    return r;
}
```

9a What is chained hashing? Discuss its pros and cons. Construct the hash table to insert the keys: 7, 24, 18, 52, 36, 54, 11, 23 in a chained hash table of 9 memory locations. Use $h(k) = k \bmod m$

Chained hashing, also known as separate chaining, is a technique used to resolve collisions in hash tables. When two or more keys hash to the same index (known as a collision), chained hashing handles these collisions by maintaining a linked list of all elements that hash to the same index.

Pros of Chained Hashing:

- 1.Simple Implementation: Chained hashing is relatively easy to implement.
- 2.Efficient Insertion and Deletion: Insertion and deletion operations are efficient in chained hashing.
- 3.Dynamic Data Sets: Chained hashing allows for dynamic resizing of the hash table..
- 4.Adaptability: Chained hashing can handle a wide range of input data distributions

Cons of Chained Hashing:

- 1.Memory Overhead: Chained hashing can have a higher memory overhead compared to other collision resolution techniques.
- 2.Cache Inefficiency: cache inefficiency, particularly for large hash tables with long linked lists.
- 3.Performance Degradation:
- 4.Poor Worst-Case Performance: Chained hashing does not guarantee constant-time performance for lookup operations in the worst case

Solution:

9a) 7, 24, 18, 52, 36, 54, 11, 23

table size 9 & An element can be mapped to location using $\text{Key} \times 9$

Hash Table Location

Hash Table Location	Mapped Element
0	18, 36, 54
1	
2	11
3	
4	
5	52
6	24
7	7, 52
8	

1) $7 \times 9 = 7$ 2) $24 \times 9 = 6$ 3) $18 \times 9 = 0$

4) $52 \times 9 = 7$ (collision occurred)

5) $36 \times 9 = 0$ (collision occurred)

6) $54 \times 9 = 0$ (collision occurred)

7) $11 \times 9 = 2$

8) $23 \times 9 = 5$

Hash Table

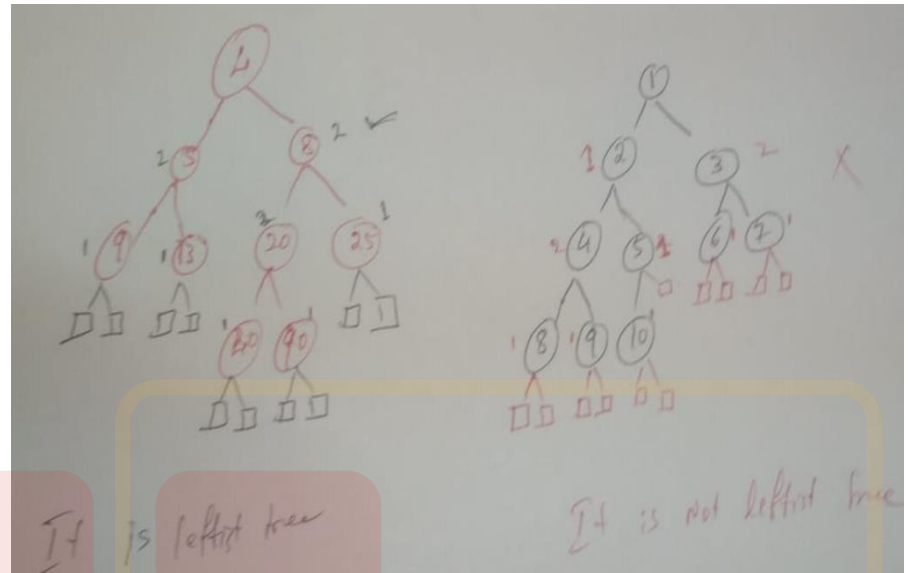
0	→ 18 → 36 → 54
1	
2	→ 11
3	
4	
5	→ 52
6	→ 24
7	→ 7 → 52
8	

9 b) Define the leftist tree. Give its declaration in C. Check whether the given binary tree is a leftist tree or not. Explain your answer.

A leftist tree is a binary tree such that if it is not empty, then
 $\text{shortest}(\text{LeftChild}(x)) \geq \text{shortest}(\text{RightChild}(x))$ for every internal node x .

C declaration

```
struct node
{
    int data;
    struct node *left;
    struct node *right;
    int rank;
};
```



9 c) What is dynamic hashing? Explain the following techniques with examples: i) Dynamic hashing using directories ii) Directory less dynamic hashing.

Dynamic hashing is a mechanism for dynamically adding and removing data buckets on demand.

Dynamic hashing using directories:

- ✓ It is a method of hashing in which the data structure grows and shrinks
- ✓ Dynamically as records are added or removed.
- ✓ It is also known as extendible hashing.
- ✓ This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

Buckets: The buckets are used to hash the actual data.

Directories: They are the holders of pointers pointing towards these buckets. Each directory has a unique ID.

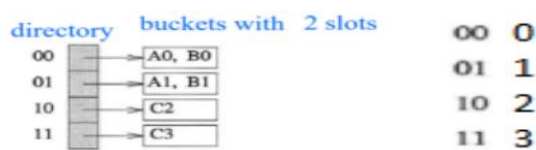
Global Depth: They denote the number of bits which are used by the hash function to categorize the keys.

Local Depth: It is associated with the buckets.

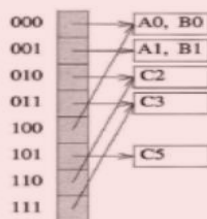
Working:

- Initialize the bucket depths and the global depth of the directories.
- Convert data into a binary representation.
- Consider the "global depth" number of the least significant bits (LSBs) of data.
- Map the data according to the ID of a directory.
- Check if a bucket overflows or not.

Example 2: Given A0, A1, B0, B1, C1, C2, C3 **Depth =2** convert given numbers into binary digits.



Example 3: Given A0, A1, B0, B1, C1, C2, C3, C5 **Depth =3** convert given numbers into binary digits.



Directory less dynamic hashing

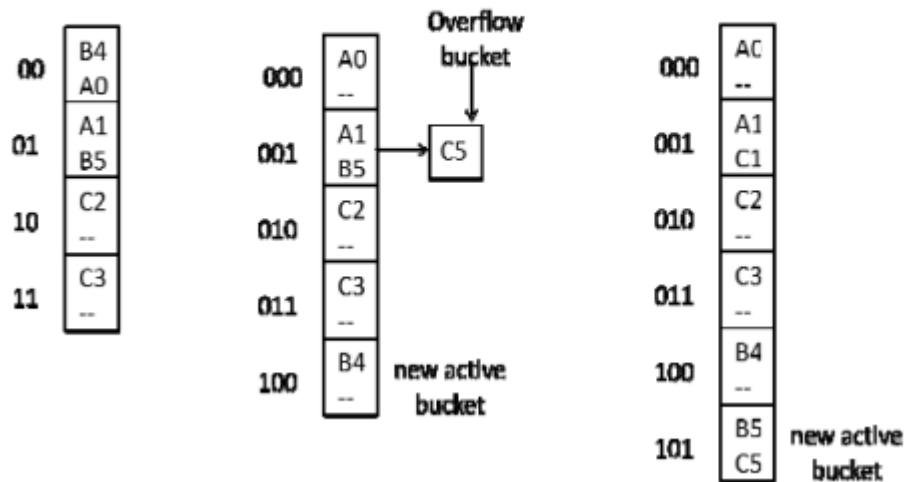
Also known as linear dynamic hashing.

Directory less hashing assume a continuous address space in the memory to hold all the records.

Therefore, the directory is not needed.

Thus, the hash function must produce the actual address of a page containing the key.

Each active bucket has 2 slots. Insert B4, A0, A1, B5, C2 and C3.



10 a) What is a Priority queue? Demonstrate functions in C to implement the Max Priority queue with an example. i) Insert into the Max priority queue ii) Delete into the Max priority queue iii) Display Max priority queue

It is a collection of ordered elements that provides fast access to the minimum or maximum element.

i) Insert into the Max priority

```
void insert_max_heap(element item, int *n)
{
    /*insert item into a max heap of current size *n */
    int i;
    if (HEAP_FULL(*n)){
        fprintf(stderr, "The heap is full. \n");
        exit(1);
    }
    i = ++(*n);
    while ((i != 1) && (item.key > heap[i/2].key)) {
        heap[i] = heap[i/2];
        i /= 2;
    }
    heap[i] = item;
}
```

ii) Delete into the Max priority.

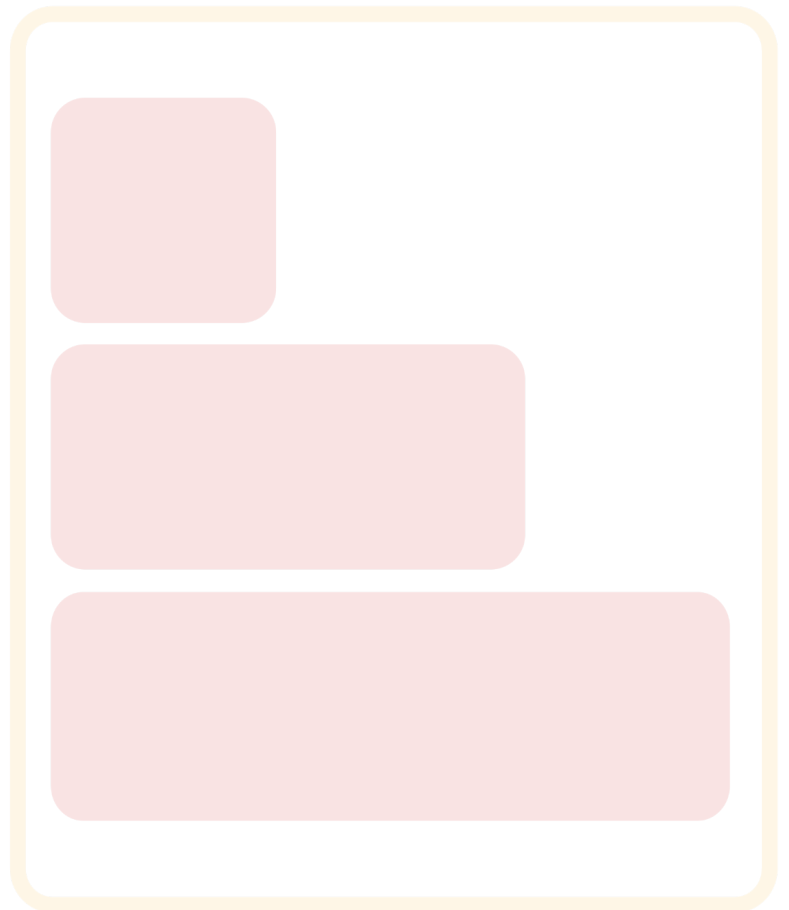
```
element delete_max_heap(int *n)
{
    /* delete element with the highest key from the heap */
    int parent, child;
    element item, temp;
    if (HEAP_EMPTY(*n)) {
        fprintf(stderr, "The heap is empty\n");
        exit(1);
    }
    /* save value of the element with the highest key */
    item = heap[1];
    /* use last element in heap to adjust heap */
    temp = heap[(*n)--];
    parent = 1;
    child = 2;
    while (child <= *n) {
        /* find the larger child of the current parent */
        if (child < *n && (heap[child].key < heap[child+1].key))
            child++;
        if (temp.key >= heap[child].key) break;
        /* move to the next lower level */
        heap[parent] = heap[child];
        parent = child;
    }
    heap[parent] = temp;
}
```

iii) Display Max priority queue

```
void display_pqueue()
{
    if ((front == -1) && (rear == -1))
    {
        printf("\nQueue is empty");
        return;
    }

    for (; front <= rear; front++)
    {
        printf(" %d ", pri_que[front]);
    }

    front = 0;
}
```

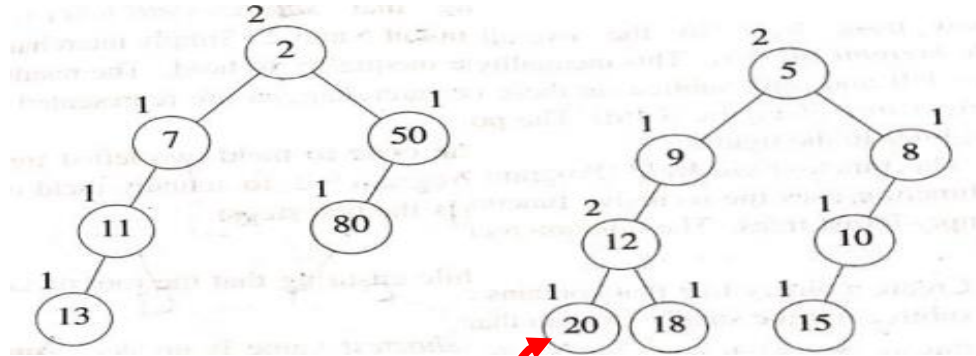


10 b) Define min Leftist tree. Meld the given min leftist trees.

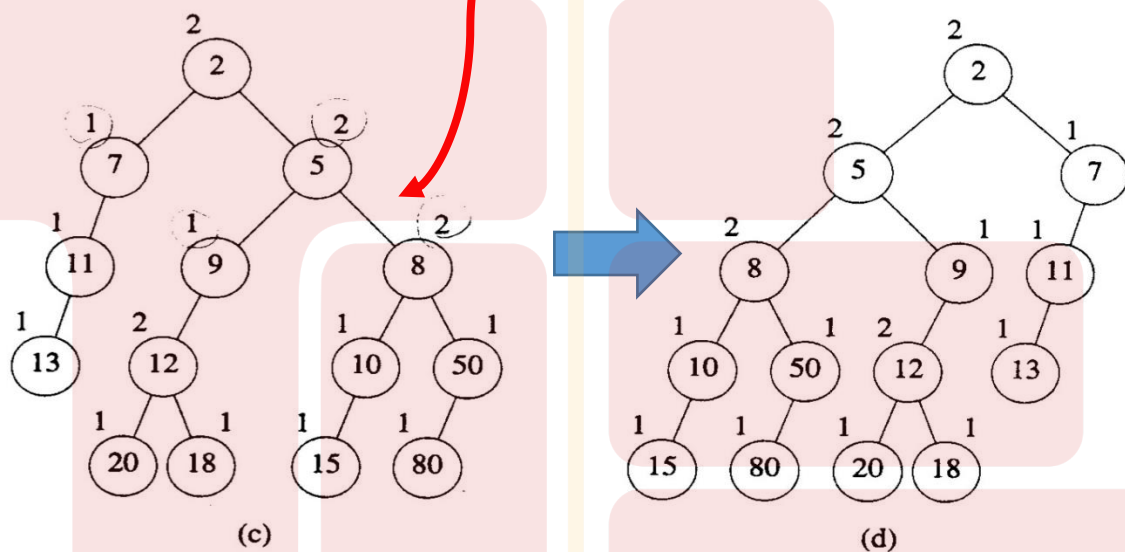
Definition:

A min-leftist tree (max leftist tree) is a leftist tree in which the key value in each node is no larger (smaller) than the key values in its children.

Given :



Melding



Hint: meld means joining two leftist tree into single leftist tree

10 c) Define hashing. Explain different hashing functions with examples. Discuss the properties of a good hash function.

A function that converts a given big number to a small integer value. The mapped integer value is used as an index in the hash table.

$$\text{hash Location} = \text{KEY} \% \text{SIZE}$$

Types of Hash Functions-

- Mid Square Hash Function
- Division Hash Function
- Folding Hash Function.
- Digit analysis
- Converting keys to integers

Division Method In this method we use modular arithmetic system to divide the key value by some integer divisor m (table size).

$x = 23, m = 10$ then

$$H(x) = (23 \% 10) = 3$$

The key whose value is 23 is placed in 3rd location

0	
1	
2	
3	23
4	

Midsquare Methods

In this case, we square the value of a key and take the number of digits required to form an address, from the middle position of squared value.

Suppose a key value is 16, then its square is 256.

Now if we want address of two digits, then we select the address as 56 (i.e. two digits starting from middle of 256).

Folding Method

The key is actually partitioned into number of parts, each part having the same length as that of the required address.

This is done in two ways :

- ☐ Fold-shifting
- ☐ Fold-boundary

Fold-shifting: Here actual values of each parts of key are added

Example:

The key is : 12345678, and the required address is of two digits,

Then break the key into: 12, 34, 56, 78.

Add these, we get $12 + 34 + 56 + 78 = 180$

Ignore first 1 we get 80 as location (because its two digits)

Fold-boundary: Here the reversed values of outer parts of key are added.

Example:

The key is : 12345678, and the required address is of two digits,

Then break the key into: □ 12 34 56 78

Reverse the keys □ 21, 34, 56, 87.

Add these, we get $21 + 34 + 56 + 87 = 198$, ignore first 1 we get 98 as location (Bcz digits is two)

Digit Analysis

Here we make a statistical analysis of digits of the key, and select those digits (of fixed position) which occur quite frequently.

For example, if the key is : 9861234.

Here third and fifth position digits occur quite frequently, then we choose the digits in these positions from the key.

So we get, 62.

Reversing it we get 26 as the address