CBCS SCHEME

USN | | | | | | | | | | |

BCS304

# Third Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025
## Data Structures and Applications

Time: 3 hrs. Max. Marks: 100

*Note: 1. Answer any FIVE full questions, choosing ONE f.. question from each module.*
*2. M : Marks , L: Bloom's level , C: Course outcomes.*

| | | Module – 1 | M | L | C |
|---|---|---|---|---|---|
| Q.1 | a. | Define Data Structures. Explain the classification of data structures with a neat diagram. | 8 | L2 | CO1 |
| | b. | Write a C Functions to implement pop , push and display operations for stacks using assays. | 7 | L2 | CO2 |
| | c. | Differentiate structures and unions. | 5 | L2 | CO1 |
| | | OR | | | |
| Q.2 | a. | Write an algorithm to evaluate a postfix expression and apply the same for the given postfix expression.<br>6 2 / 3 – 4 2 * +. | 7 | L3 | CO2 |
| | b. | Explain the dynamic memory allocation function in detail. | 8 | L2 | CO1 |
| | c. | What is Sparse matrix? Give the triplet form of a given matrix and find its transpose<br>$A = \begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$ | 5 | L3 | CO1 |
| | | Module – 2 | | | |
| Q.3 | a. | Define Queue. Discuss how to represent a queue using dynamic assays. | 8 | L2 | CO2 |
| | b. | Write a C Function to implement insertion ( ) , deletion ( ) and display ( ) operations on circular queue. | 6 | L3 | CO2 |
| | c. | Write a note on Multiple stacks and queues with suitable diagram. | 6 | L2 | CO2 |
| | | OR | | | |
| Q.4 | a. | What is a linked list? Explain the different types of linked list with neat diagram. | 6 | L2 | CO3 |
| | b. | Write a C function for the following on singly linked list with example :<br>i) Insert a node of the beginning<br>ii) Delete a node at the front<br>iii) Display. | 8 | L3 | CO3 |
| | c. | Write the C function to add two polynomials. | 6 | L2 | CO3 |

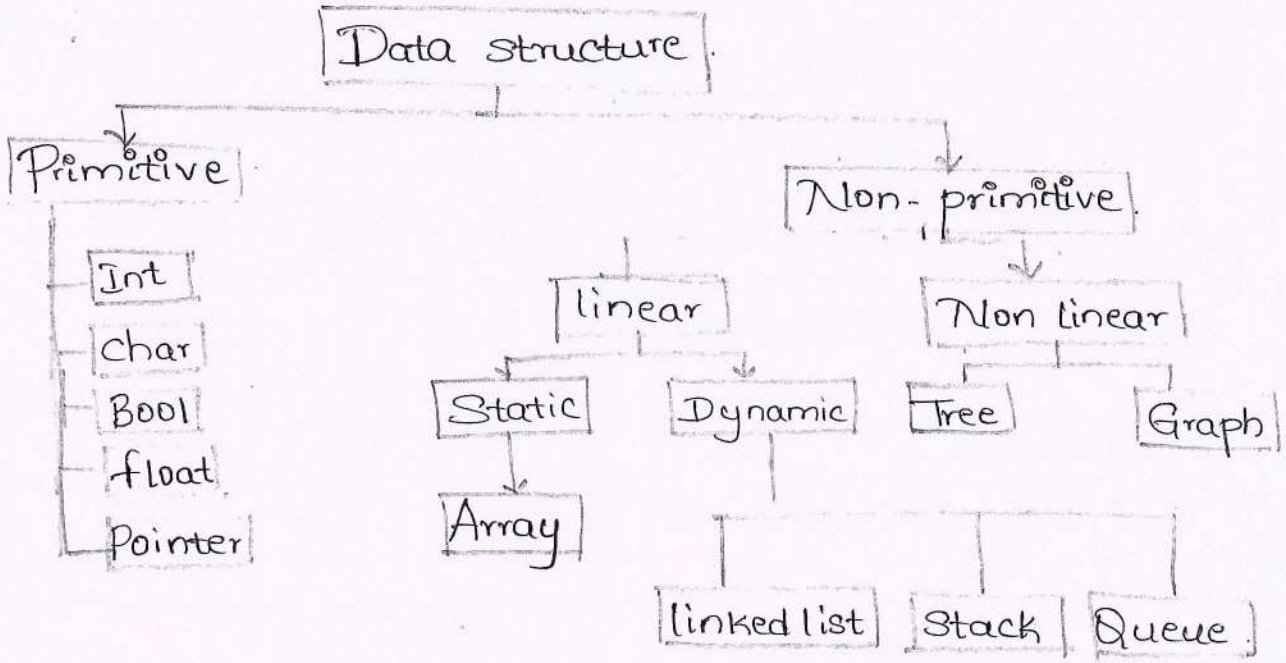
| | | | | | |
|---|---|---|---|---|---|
| | | Module – 3 | | | |
| Q.5 | a. | Discuss how binary trees are represented using : i) Assay ii) Linked list. | 6 | L2 | CO4 |
| | b. | Define Threaded binary tree. Discuss In – threaded binary tree. | 6 | L2 | CO4 |
| | c. | Write the C function for the following additional list operation :<br>i) Inverting Singly linked list ii) Concatenating Singly linked list. | 8 | L3 | CO3 |
| | | OR | | | |
| Q.6 | a. | Discuss Inorder , Preorder , Postorder and Level order traversal with suitable function for each. | 8 | L3 | CO4 |
| | b. | Define the threaded binary tree. Construct threaded binary tree for the following element : A, B, C, D, E, F, G, H, I. | 6 | L2 | CO4 |
| | c. | Write a C function for the following :<br>i) Insert a node at the beginning of doubly linked list.<br>ii) Deleting a node at the end of the doubly linked list. | 6 | L3 | CO3 |
| | | Module – 4 | | | |
| Q.7 | a. | Define Forest , Transform the forest into a binary tree and traverse using inorder , preorder and postorder traversal with an example. | 8 | L1 | CO5 |
| | b. | Define Binary search tree. Construct a binary search tree for the following elements : 100 , 85 , 45 , 55 , 120 , 20 , 70 , 90 , 115 , 65 , 130 , 145. | 6 | L2 | CO5 |
| | c. | Discuss Selection tree with an example. | 6 | L2 | CO5 |
| | | OR | | | |
| Q.8 | a. | Define Graph. Explain adjacency matrix and adjacency list representation with an example. | 8 | L2 | CO5 |
| | b. | Define the following terminology with example :<br>i) Digraph ii) Weighted graph iii) Self loop iv) Connected graph. | 6 | L2 | CO5 |
| | c. | Briefly explain about Elementary graph operations. | 6 | L3 | CO5 |
| | | Module – 5 | | | |
| Q.9 | a. | Explain in detail about Static and Dynamic Hashing. | 6 | L2 | CO5 |
| | b. | What is Collision? What are the methods to resolve collision? | 7 | L2 | CO5 |
| | c. | Explain Priority queue with the help of an examples. | 7 | L2 | CO5 |
| | | OR | | | |
| Q.10 | a. | Define Hashing. Explain different hashing functions with suitable examples. | 12 | L2 | CO5 |
| | b. | Write short note on :<br>i) Leftist trees ii) Optimal binary search tree. | 8 | L3 | CO5 |

* * * * *

# Data Structures and Applications (BCS304)

## Scheme of solution.

Q1.a. Data may be organised by different ways, the logical or mathematical model of a particular organizatio of data is called a data structure.




1. Primitive data structure: Can be manipulated directly by machine instruction called primitive data structure.
2. Non primitive data structure: Cannot be manipulated directly by machine instruction called non-primitive data structure.

Based on structure & arrangement of data.

* Linear Data structure: A Data structure is said to be linear if its elements form a sequence or linked list.
* Non linear Data Structure: A data structure is said to be non linear if its data are not arranged in sequence or linear.

* Static data Structure: are those that have a fixed size & structure at compile time.

* An array is a systematic arrangement of similar objects usually in rows & columns.

* Dynamic Data Structure: are the size of structure is not fixed and can be modified during the operations performed on it.

* Tree :- is a collection of nodes connected by directed (or undirected) edges.

* Graph :- A non-linear kind of data structure made up of nodes or vertices & edges.

* Linked list: is a linear collection of data elements whose order is not given by their physical placement in memory.

* Stack: is a linear list in which insertion & deletions can take place only at one end called top.

* Queue: is a linear data structure that is open at both ends and the operations are performed in FIFO order.

8b. Push operation, pop and display operation.

```
Void Push()
{
 if (top==MAX-1)
  printf("stack overflow");
 else
  {
   printf("Enter the item to be pushed\n");
   scanf("%d", &item);
   top=top+1;
   stack[top]=item;
  }
}
```

```
Void pop()
{
  if (top==-1)
  printf("Stack underflow");
  else
   {
    item = Stack[top];
    top = top-1;
    printf("deleted item is %d",item);
   }}
```

```
Void display()
 {
   int i;
   if (top==-1)
   printf("stack is empty");
   else
    {
     for (i=top ; i>=0; i--)
     printf("%d\t", stack[i]);
   }}
```

Q1. C:

| Structure | Union |
|---|---|
| 1) The keyword struct is used to declare a structure. | 1) The keyword union is used to declare a union. |
| 2) Individual member can be accessed at a time. | 2) Only one member can be accessed at a time. |
| 3) Structure member don't share their memory. | 3) Union members share by memory for other union members. |

Q2.a) Step 1 : Start

Step 2 : Check all symbols from left to right & repeat Step 3 & 4 for each symbol of expression 'E' until all symbols are over. i) If the symbol is an operand push it onto a stack.

ii) If the symbol is an operator then.

iii) Pop the top 2 operands from the stack & apply an operator in between them.

iv) Evaluate the expression & place the result back on the stack.

Step 3 : Set the result equal to a top element on the stack,

Step 4 : Stop.

Evaluating postfix expression.

6 2 / 3 - 4 2 * +

3 ∴ 6/2 = 3

3 3 - 4 2 * +

0

0 4 2 * + ∴ 3 - 3 = 0

8 + 4 * 2 = 8

∴ 6 2 / 3 - 4 2 * + = 8.

2b. Dynamic memory allocation

* Many language permit a programmer to specify an array's size at runtime.

* Such language the ability to calculate & assign during execution, the memory space required by variables in a program. The process of allocating memory at runtime Called dynamic memory allocation.

1) malloc() :- A block of memory be allocated using function malloc.

* It reserves a block of memory of specified size & return a pointer of type void.

```
ptr = (type*) malloc(byte-size);
```

ptr is a pointer of type, malloc returns a pointer to an area of memory with size bytesize.

```
x (int*) malloc (100 * sizeof (int));
```

A memory space equivalent to "100 times the size of an int" byte is reserved & address of 1st byte of memory allocated is assigned to point x of type int.

2) Calloc(): allocates multiple block of storage each of the same size, & sets all bytes to zero.

```
ptr = (cast type*) calloc (n, element_size);
```

* It allocates contiguous space for n blocks, each of size elmesize bytes.
* All bytes an initialized to zero & a pointer to 1st byte of allocated region is returned.

* If there is not enough space, a Null pointer is returned.

3) Free(); When no longer need the data we stored in a block of memory.

* We donot intend to use that block for storing any other informatn using free functn().

free(ptr);

ptr is a pointer to a memory block which are already created by malloc or calloc.

4) Realloc(); We can change the memory size already allocated with help of the function realloc.

ptr = malloc(size);

then reallocation of space may be done by

ptr = realloc(ptr, newsize);

This function allocates a new memory space for size, newsize to the pointer variable ptr and returns a pointer to the first byte of new memory block.

2c. A matrix which contain many zero entries or very few non-zero entries called as sparse matrix.

$$A=\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Triplet form

| row | col | value |
|---|---|---|
| 3 | 4 | 6 |
| 0 | 2 | 3 |
| 0 | 4 | 4 |
| 1 | 2 | 5 |
| 1 | 3 | 7 |
| 3 | 1 | 2 |
| 3 | 2 | 6 |

Interchanging rows & columns

| | | |
|---|---|---|
| 2 | 0 | 3 |
| 4 | 0 | 4 |
| 2 | 1 | 5 |
| 3 | 1 | 7 |
| 1 | 3 | 2 |
| 2 | 3 | 6 |

transpose of Sparse matrix

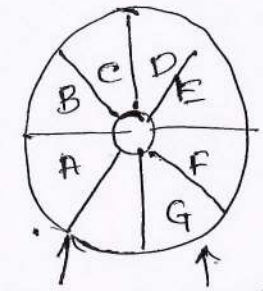
$$\begin{bmatrix} 1 & 3 & 2 \\ 2 & 0 & 5 \\ 2 & 1 & 5 \\ 2 & 3 & 6 \\ 3 & 1 & 7 \\ 4 & 0 & 4 \end{bmatrix} \ldots\ldots$$

30. a) A queue is an ordered list in which insertion & deletion take place at different ends.

Queue representation using dynamic array.

* A dynamically allocated array is used to hold the queue elements. Let capacity be no. of position in array queue
* To add an element to full queue, 1st increase the size of this array using a function realloc, dynamically allocated stacks, array doubling is used.
* Consider, the full queue in (a) shows a queue with 7 elements in an array whose capacity is 8.

A circular queue is flatten out the array in (b)




(a) A full circular queue.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|---|---|---|---|---|---|---|---|
| C | D | E | F | G | | A | B |

front = 5 rear = 4

(b) flattened view of circular full queue.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | D | E | F | G | | A | B | | | | | | | | |

front = 5 rear = 4

(c) After array doubling

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | D | E | F | G | | | | | | | | | | A | B |

front = 3 rear = 4

(d) After shifting right segment.

fig(e) follows the steps

1) Create a new array newQueue of twice the capacity.
2) COPY the 2nd segment to position in newQueue beginning at 0.
3) Copy the 1st segment to position in newQueue beginning at capacity - front - 1.

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | | | | | | | | | |

front = 15 rear = 6

(e) Alternative configuration.

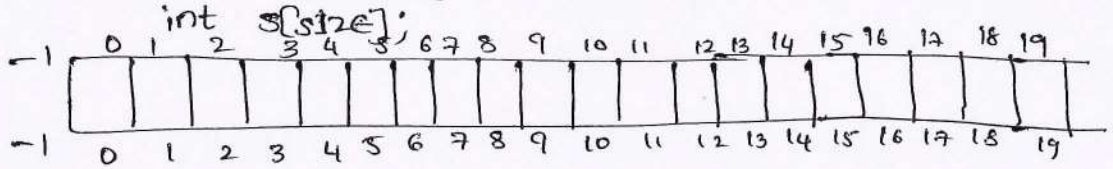
3b. C function to implement insertion(), deletion() & display() operation.

```
Void insert()
{
 if (count == MAX)
  printf("Queue Overflow");
 else
 {printf("Enter the item to be inserted\n");
  Scanf("%c", &item);
   r=(r+1)%MAX
   q[r]=item;
    count++;
}}
Void delete()
{
 if (count==0)
  Printf("Queue underflow\n");
 else
 {printf("Deleted item %c\n", q[f]);
  f=(f+1)%MAX;
   count--;
}}
Void display()
{
 int j=f, i;
 if (count==0)
  Printf("queue is empty\n");
 else
  {
  printf("Contents of circular queue\n");
  for(i=1; i<=count; i++)
   {
```

```
        printf("%c\t", q[j]);
        j = (j+1) % MAX;
    }
    printf("total no. of items = %d\n", count);
}
```

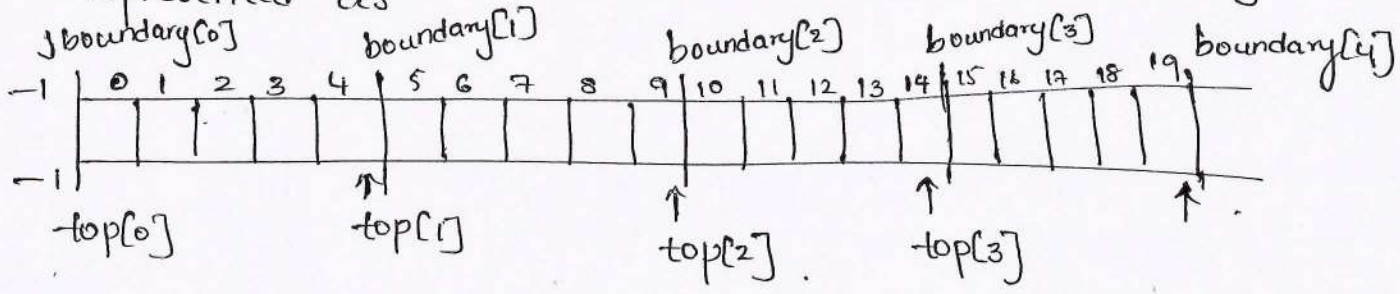
3c. Let us implement multiple stacks using array.

```
#define size 20
int s[size];
```




* Now let us divide the array into 'n' stacks

Ex: if n=4 then 4 empty stacks can be pictorially represented as




* To find the initial value of top of each stack i.e top[0], top[1], top[2], top[3], use the expression

top[j] = size/n * j - 1 where j = 0 to n

i.e j=0, top[0] = 20/4 * 0 - 1 = -1

j=1, top[1] = 20/4 * 1 - 1 = 04

j=2, top[2] = 20/4 * 2 - 1 = 9

j=3, top[3] = 20/4 * 3 - 1 = 14

To find the boundary of each stack, copy initial value of top[0] to boundary[0] & so on.

i.e
```
for(j=0; j<=n; j++)
{
    boundary[j] = top[j];
}
```
OR

boundary[j] = top[j] = size/n * j - 1;

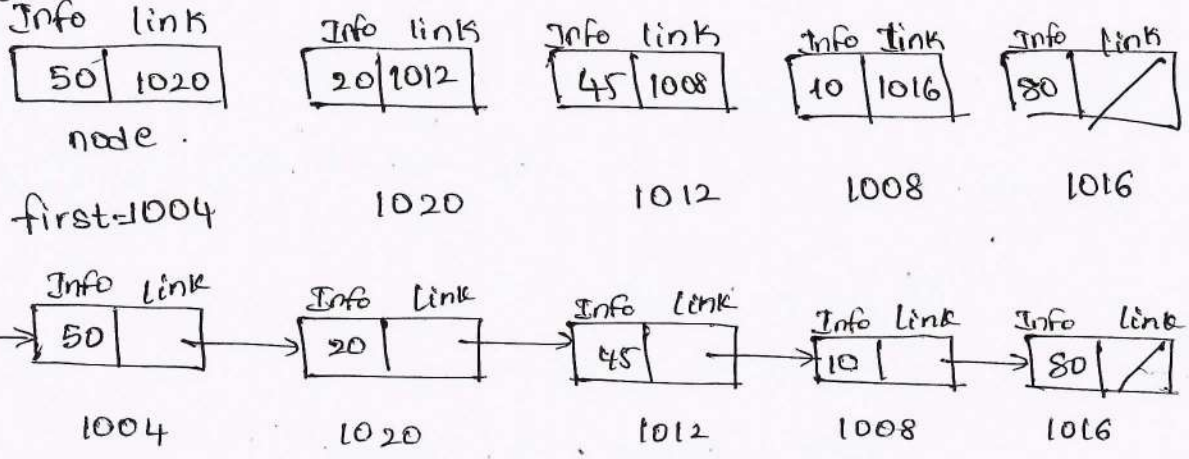
4a. A linked list is a data structure which is collection of zero or more nodes where each node has some information.

Types of linked list.

1) Singly linked list.
2) Doubly linked list.
3) Circular listked list

1) Singly linked list: is a linked lists where each node has designated field called link field which contain address of next node. If there exists only one link field in each & every node in the list, called singly linked list.

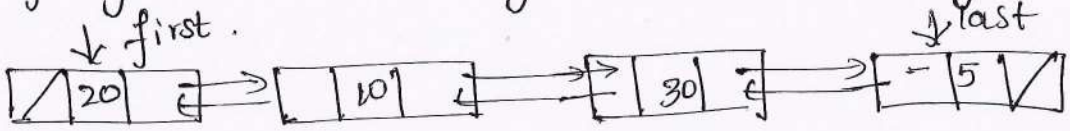
Ex:



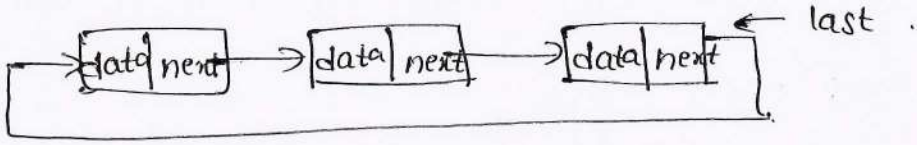

2) Doubly linked list

A list where both forward & backward traversal is possible should have 2 link fields. The link field which contain address of the left node is called left link denoted by llink & link field which contain the address of right node called right link. denoted by rlink.

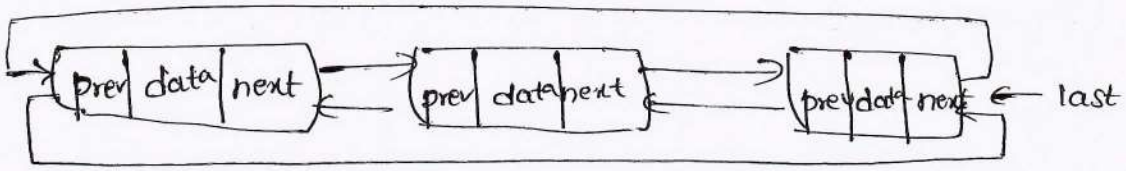
Ex:



3) Circular Doubly / Singly linked list.




Circular singly linked list, each node has just one pointer called the 'next' pointer. The next pointer of last node points back to first node & result in circular fashion.




Circular doubly linked list.

4 b.i) Insert a node of the beginning.

```
Void insert_front(listpointer *first, listpointer x).
{
  if (First=NULL)
    listpointer temp;
    MALLOC(temp, sizeof(*temp));
     if (*first) {
       temp→link = x→link;
        x→link = temp;
      }
    else {
        temp→link = NULL;
         *first = temp;
}}
```

ii) Delete a node at the front.

```
Void delete_front()
{
 p=first;
 if(first==NULL)
 {printf("list is empty");
 }
 else if(p->link==NULL)
 {
  printf("deleted node '%d'\n");
   free(p);
   first=NULL;
    count--;
  }
 else
    {
     first=p->link;
     Printf("delete node \n");
      free(p);
       count--;
    }}
```

iii) Display

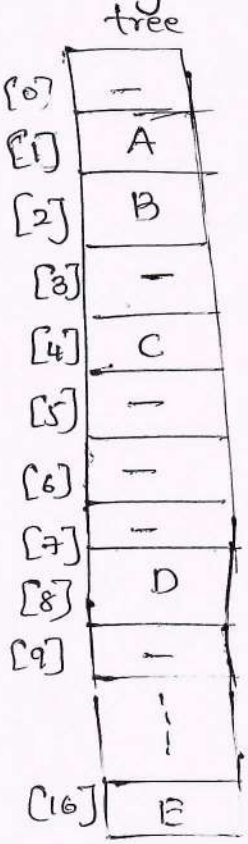
```
Void display()
 {
   if(first==NULL)
    {printf("list is empty\n");
   }else
   {p=first;
     printf("content of list \n");
      while(p!=NULL)
```

```
        printf(" Display");
        p = p -> link;
    }
}
```

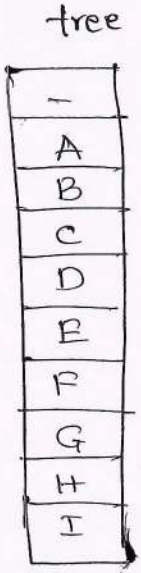
4c.
```
Polypointer cpadd(polypointer a, polypointer b)
{
    Polypointer startA, c, lastc;
    int sum, done = FALSE;
    startA = a;
    a = a -> link;
    b = b -> link;
    c = getNode();
    c -> expon = -1;
    lastc = c;
    do {
        Switch (COMPARE (a -> expon, b -> expon)) {
        case -1: attach (b -> coef, b -> expon, &lastc);
                 b = b -> link;
                 break;
        case 0:  if (startA == a)
                     done = TRUE;
                 else {
                     Sum = a -> coef + b -> coef;
                     if (sum)
                     attach (sum, a -> expon, &lastc);
                     a = a -> link;
                     b = b -> link;
                 }
                 break;
```

```
        case 1: attach (a→coef, a→expon, &lastC);
                a = a→link;
            }
        } while (!done);
        lastC → link = C;
            return C;
        }
```

5 a. i) Array Representation.




(a)




(b)

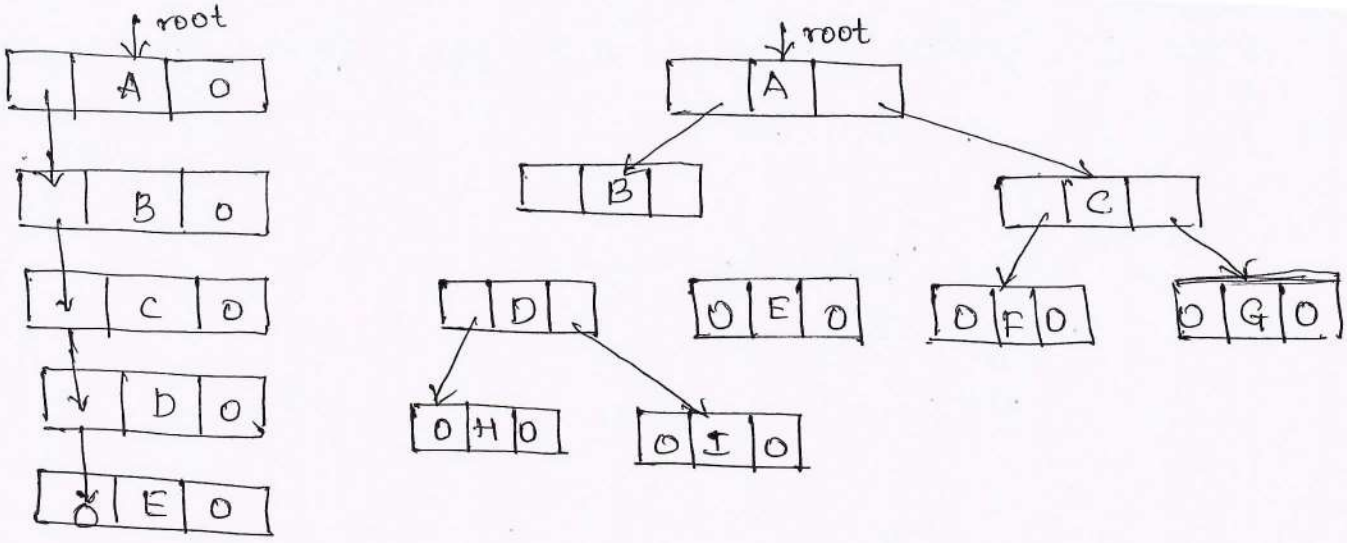
(a) Skewed tree less than half the array is ultized.

(b) Complete binary tree, the representation is ideal.

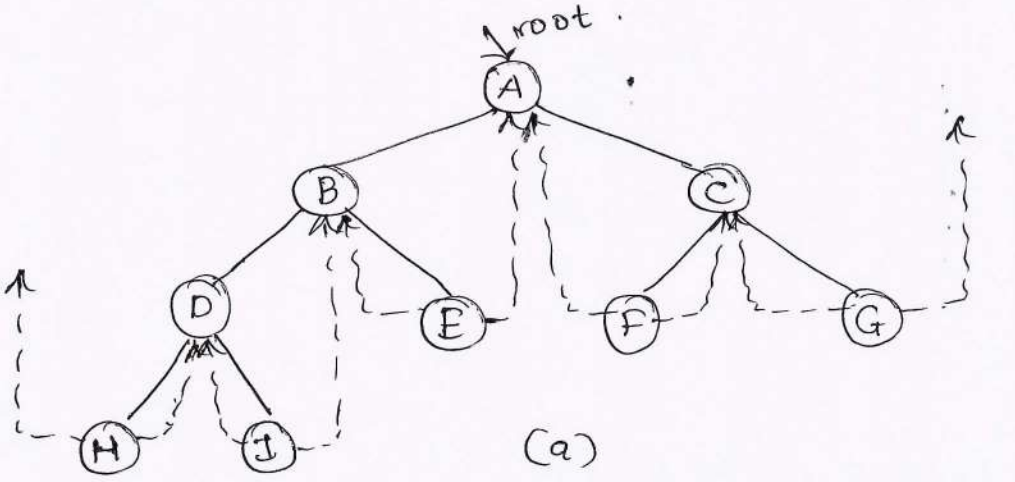
In Skewed tree of depth k will require $2^k - 1$ space otherwise only k will be used.

ii) Linked list. Representation

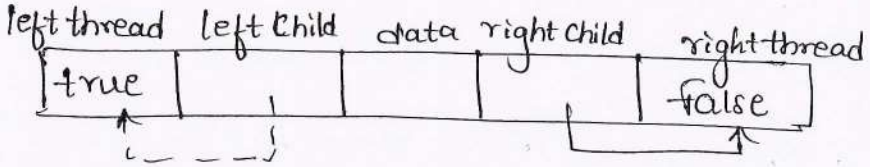
The root of the tree is stored in the data member root of Tree. This data member serves as the access the pointer to the tree.




b. A threaded binary tree is a type of binary tree data structure where the empty left & right child pointers in a binary tree are replaced by with threads that link nodes directly to their inorder predecessor or successor, thereby providing a way to traverse the tree without using recursion or a stack.




(a)




(b) empty threaded binary tree.

(a) One in left child H, the other right child of G. In order we leave no loose threads.

(b) Empty threaded binary tree is represented by its header node.

c. i) Inverting singly linked list.

```
listPointer invert (listpointer lead)
{
  listpointer middle, trail;
  middle = NULL;
  while (lead) {
    trail = middle;
    middle = lead;
    lead = lead → link;
    middle → link = trail;
  }
  return middle;
}.
```

ii) Concatenating linked list.

```
listpointer concatenate (list pointer ptr1, listpointer ptr2)
{
  listpointer temp;
  if (!ptr1) return ptr2;
  if (!ptr2) return ptr1;
  for (temp = ptr1; temp → link; temp = temp → link)
    temp → link = ptr2;
}.
```

6a. 

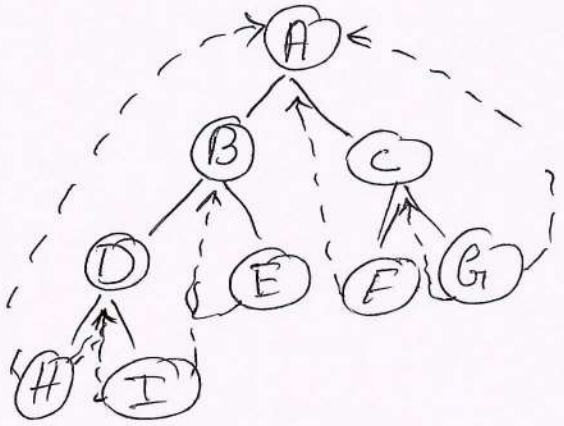
```
void inorder (treepointer ptr)
{
  if (ptr) {
    inorder (ptr → leftchild);
    printf ("%d", ptr → data);
    inorder (ptr → rightchild);
  }
}
```

```
Void preporder (treepointer ptr)
{
    if (ptr) {
        printf ("%d", ptr->data);
        preorder (ptr->leftchild);
        preorder(ptr->rightchild);
    }}

Void postorder (treepointer ptr)
{
    if (ptr) {
        postorder (ptr->leftchild);
        postorder (ptr->rightchild);
        printf ("%d", ptr->data);
    }}

Void levelorder (treepointer ptr)
{
    int front = rear=0;
    treepointer queue [MAX_QUEUE_SIZE];
    if (!ptr)
        return;
    addq(ptr);
    for(;;) {
        ptr = deleteq();
        if (ptr) {
            printf("%d", ptr->data);
            if (ptr->leftchild)
                addq(ptr->leftchild);
            if(ptr->rightchild)
                addq(ptr->rightchild);
        } else break;
```

6b) In linked representation of any binary tree there are more null links than actual pointers. Specifically there are n+1 null links out of 2n total links. Replacing these null links by pointers called threads to other nodes in the tree.



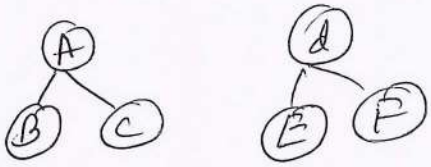

6c) Insert node at the begining of DLL

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

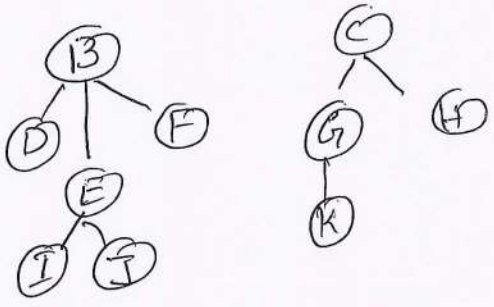
ii) Delete the End of DLL

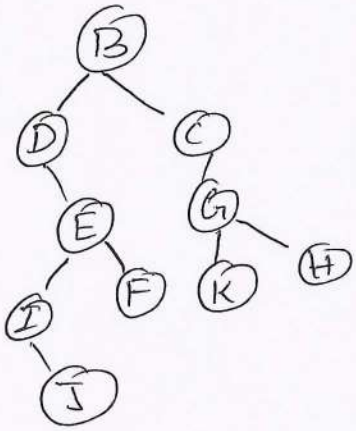
```
void deleteEnd()
{
    tmp = head;
    while(tmp->right != NULL)
    {
        tmp = tmp->right;
    }
    last = temp->right;
    last->right = NULL;
    tmp->left = NULL;
    printf("Node is deleted:%d", tmp->data);
    free(tmp);
}
```

7a) A forest is a set of $n > 0$ disjoint trees.




eg:-



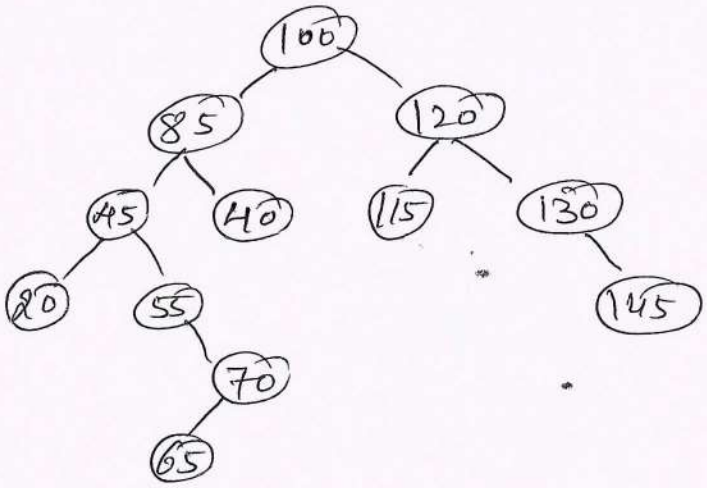

Inorder traversal → D I J E F B K G H C

Preorder traversal → B D E I J F C G K H

Postorder traversal → J I F E D K H G C B

7b) Binary Search tree :- Every elements has a unique key. The keys in non empty left subtree are smaller than the key in the root and right subtree are larger than the key in the root of subtree.

100, 85, 45, 55, 120, 20, 70, 90, 115, 65, 130, 145

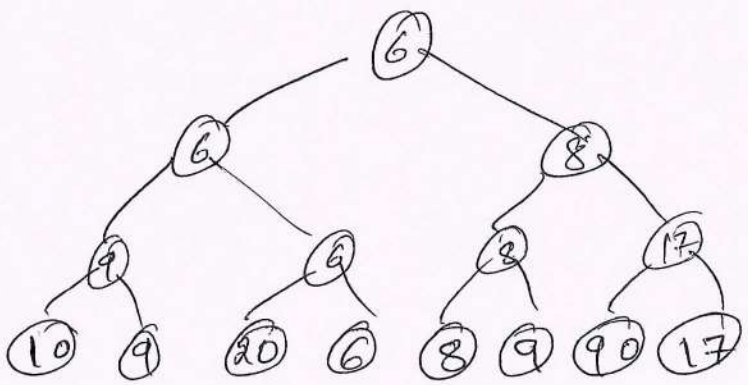



7c) Selection tree :- It is also called as tournament tree This is such a tree data structure using which the winner of a knock out tournament can be selected.

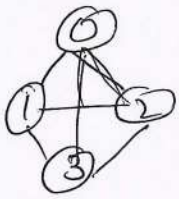
There are two types of selection trees

* Winner Tree :- It 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.

* Looser Trees :- In which each non-leaf node retains a pointer to the loser.

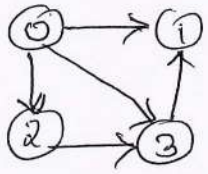


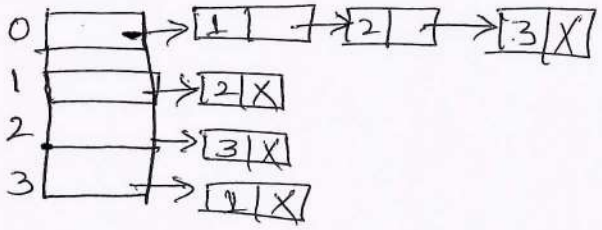
8a) Graph is a pair of sets $(V, E)$, where $V$ is the set of vertices & $E$ is the set of edges connecting the pairs of vertices. $G = (V, E)$




Adjacency List :- It is a way in which graphs can be represented in the array of 'n' linked lists.

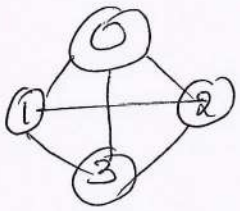
eg :-




Adjacency matrix :- Let $G = (V, E)$ be graph. Let $n$ be the number of vertices in graph $G$. The adjacency matrix of a graph $g$ is

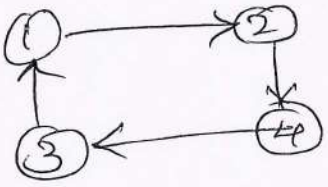
$$A[i][j] = \begin{cases} 1 & \text{if there is an edge exists} \\ 0 & \text{if there is no edge.} \end{cases}$$



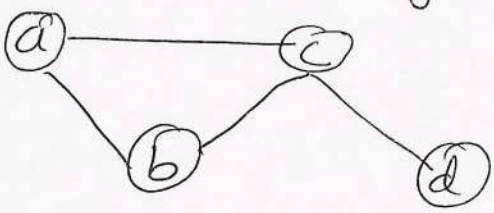

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 | 0 |

8b) Digraph :- It is also known as directed graph it contains directed edges to the vertices

eg :-

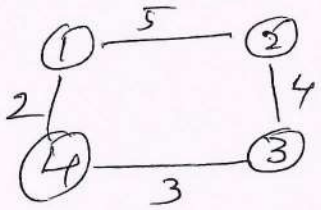


Connected graph :- It is a graph in which there is a path between every pair of vertices.




Weighted graph :- All edges have a weight typically shows cost of traversing.

eg :-




selfloop :- It is an edge which starts & ends are the same vertex


8c) Graph operations :-

BFS uses queue data structure

Step1 :- Visit the adjacent unvisited vertex. Mark it as visited. Display it. And insert in a queue.

Step2 :- If no adjacent vertex is found, remove the first vertex from the queue.

Step3 :- Repeat Rule1 & Rule2 until the queue is empty.

DFS - uses stack data structure.

Step1 :- Visit the adjacent unvisited vertex. Mark it as visited. Display it, push it on a stack.

Step2 :- If no adjacent vertex is found, pop up a vertex from the stack.

Step3 :- Repeat Rule 1 & Rule 2 until the stack is empty.

9a) Static hashing :- When a search-key value is provided the hash function always computes the same address. It is used in database

eg :- 76 % 5 = 1

It always result in the same bucket address 1.

There will not be any changes to the bucket address here.

number buckets is constant in the memory.

Dynamic hashing :- It is a method of hashing in which the data structure grows & shrinks dynamically as records are added or removed. Also known as extendible hashing.

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

Directories :- They are the holders of pointers pointing towards these buckets

Global depth :- They denote the number of bits which are used by the hash function

Local depth :- It is associated with the buckets

96) Collision resolution is the process of handling situations where two or more key hash to the same index in a hash table.

→ seperate chaining

→ open Addressing

* Each bucket in the hash table is implemented as a linked list.

→ when a collision occurs, the new key is inserted into the linked list at the corresponding index

→ It requires additional memory for the linked lists.

* All elements are stored within the hash table itself

– when a collision occurs, algorithm searches for the next available slot in the table.

→ Techniques include linear probing, Quadratic Probing & Double hashing.

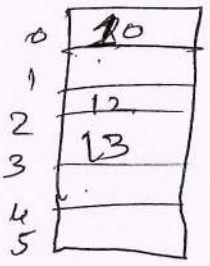
eg:- keys 12, 13, 10, 20, & table size is 10.

$h'(k,i) = [h(k) + i] \bmod n.$

$h'(20,0) = [20 \bmod 10 + i] \bmod 10$

$[0+0] \% 10$

$= 0$




here collision occurs. Now use

$i = 1,$

$[0+1] \% 10$

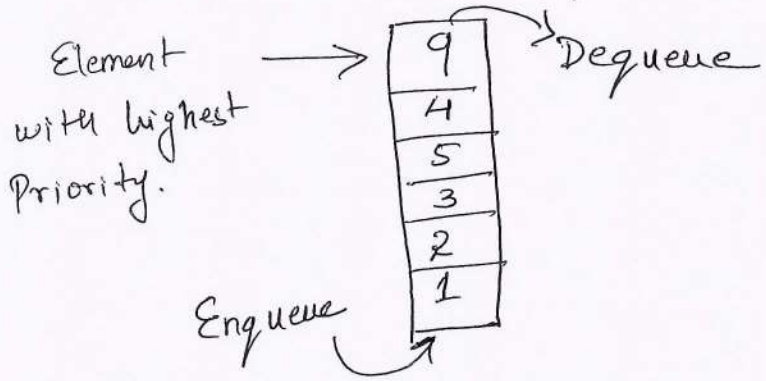
$1 \% 10$

$= 1$

In 1st location insert 20.

9c) A priority queue is a special type of queue in which each element is associated with a priority value. And elements are served on the basis of their priority.

- The element with the highest value is considered the highest priority element.
- we can also set priorities according to our needs.




- It can be implemented using an array, linked list heap data structure or binary search tree.

10a) A function that converts a given number to small integer value. The mapped integer value is used as an index in the hash table.

Address = hash (key)

H location = key % tablesize

* Hash functions :-

Division Method :- In this method we use modular arithmatic system to divide the key value by some integer divisor m (tablesize)

eg:- $x = 23$, $m = 10$ then

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

$= 3$

The given key 23 is stored on the location 3rd place.

* Midsquare Method :- 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.

eg:- Key value is 16 then square is 256.

If we want 2 digits then ignore 2, 16 is located on 5th position.

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

eg:- 12345678 & required address is of two digits.

then break the key into 12, 34, 56, 78

Add these $12+34+56+78 = 180$

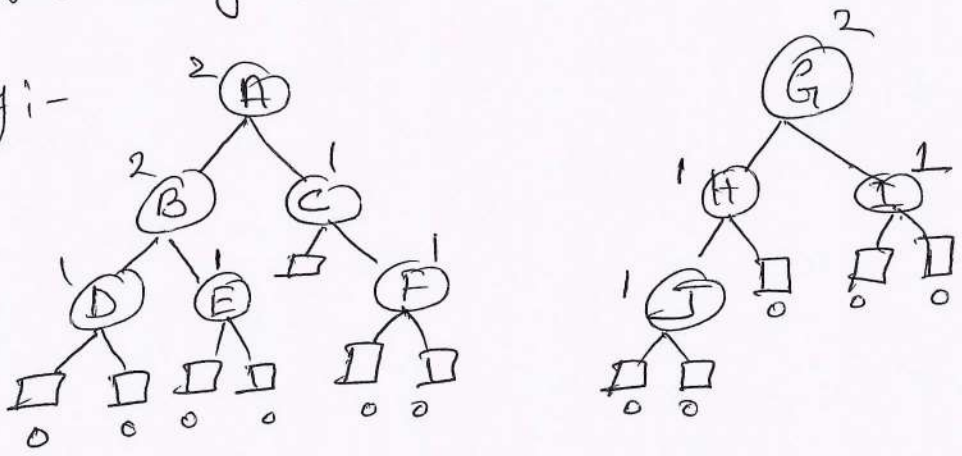
ignore 1 then we will get 80.

* Digit Analysis :- Here we make statistical analysis of digits of the key. & select those digits which occur quite frequently.

10b) Leftist tree :- It is defined using the concept of an extended binary tree. It is a binary tree in which all empty binary subtrees have been replaced by square node.

A leftist tree is a binary tree such that if it is not empty, then shortest (leftchild(x)) >= shortest(rightchild(x)) for every internal node x.

eg :-




* Optimal Binary Search Trees :-

- It is a variant of binary search trees where the arrangement of nodes is strategically optimized to minimize the cost of searches.

step1 :- Read n symbols with probability $P_i$

step2 : Create table $C(i,j)$ $1 \le i \le j+1 \le n$

step3 : Set $C[i,j] = P_i$, & $C[i-1,j] = 0$ for all $i \in (n)$

step4 : Recursively compute the following relation.

$$C[i,j] = C[1 \ldots k+1] + C[k+1 \ldots j] + \sum_{n=1}^{m} p_m \quad \text{for all } i \& j$$

Step 5: Return $C[1 \ldots n]$ as the maximum cost of Constructing BST.

Step 6: End.

HOD
Computer Science & Engineering
KLS Vishwanathrao Deshpande
Institute of Technology, Haliyal.