CBCS SCHEME

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

18CS32

Third Semester B.E. Degree Examination, Feb./Mar. 2022

Data Structures and Applications

Time: 3 hrs. Max. Marks: 100

Note: *Answer any FIVE full questions, choosing ONE full question from each module.*

Module-1

1 a. Define Data Structures. Explain the various operations on Data structures. **(06 Marks)**
b. Define Structures. Explain the types of structures with examples for each. **(07 Marks)**
c. List and explain the functions supported in C for Dynamic Memory Allocation. **(07 Marks)**

OR

2 a. Define Pattern Matching. Write the Knuth Morris Pratt Pattern matching algorithm and apply the same to search the pattern 'abcdabcy' in the text 'abcxabcdabxabcdabcy'. **(10 Marks)**
b. Write the Fast Transpose algorithm to transpose the given Sparse Matrix. Express the given Sparse Matrix as triplets and find its transpose.

$$A = \begin{bmatrix} 10 & 0 & 0 & 25 & 0 \\ 0 & 23 & 0 & 0 & 45 \\ 0 & 0 & 0 & 0 & 32 \\ 42 & 0 & 0 & 31 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 30 & 0 & 0 \end{bmatrix}$$ **(10 Marks)**

Module-2

3 a. Define Stacks. List and explain the various operations on stacks using arrays with stack overflow and stack underflow conditions. **(10 Marks)**
b. Write an algorithm to convert an infix expression to postfix expression and also trace the same for the expression $(a + b) * d + e/f + c$. **(10 Marks)**

OR

4 a. Define Recursion. Explain the types of recursion. Write the recursive function for
i) Factorial of a number ii) Tower of Hanoi. **(10 Marks)**
b. Give the Ackermann function and apply the same to evaluate A(1, 2). **(04 Marks)**
c. Explain the various operations on Circular queues using arrays. **(06 Marks)**

Module-3

5 a. Give the node structure of create a single linked list of integers and write the functions to perform the following operations :
i) Create a list containing three nodes with data 10, 20, 30 using front insertion.
ii) Insert a node with data 40 at the end of list.
iii) Delete a node whose data is 30.
iv) Display the list contents. **(10 Marks)**
b. Write the functions for : i) Finding the length of the list ii) Concatenate two lists
iii) Reverse a list. **(10 Marks)**

OR

6 a. Write the node representation for the linked representation of a polynomial. Explain the algorithm to add two polynomials represented as linked list. **(08 Marks)**

b. For the given Sparse matrix, write the diagrammatic linked list representation.

$$A \begin{bmatrix} 3 & 0 & 0 & 0 \\ 5 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 8 \\ 0 & 0 & 9 & 0 \end{bmatrix}.$$ **(04 Marks)**

c. List out the differences between single linked list and double linked list. Write the functions to perform following operations on double linked list :
i) Insert a node at rear end of the list ii) Delete a note at rear end of the list
iii) Search a node with a given key value. **(08 Marks)**

Module-4

7 a. Define a Tree. With suitable example explain i) Binary tree ii) Complete binary tree iii) Strict binary tree iv) Skewed binary tree. **(10 Marks)**

b. Write the routines to traverse the given tree using
i) Pre – Order traversal ii) Post – Order traversal **(06 Marks)**

c. Write the recursive search algorithm for a Binary Search tree. **(04 Marks)**

OR

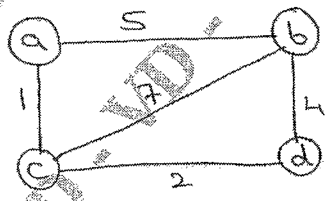
8 a. Draw a Binary tree for the following expression : ((6 + (3-2) *5) ^ 2+ 3.
Traverse the above generated tree using Pre – order and Post – order and also write their respective functions. **(10 Marks)**

b. Write the routines for :
i) Copying of binary trees ii) Testing equality of binary trees. **(10 Marks)**

Module-5

9 a. Define Graphs. Give the Adjacency matrix and Adjacency list representation for the following graph in Fig. Q9(a). **(08 Marks)**

Fig. Q9(a)



b. Write the algorithm for following Graph Traversal methods :
i) Breadth first search ii) Depth first search. **(08 Marks)**

c. Write an algorithm for insertion sort. **(04 Marks)**

OR

10 a. Define Hashing. Explain any three Hash functions. **(08 Marks)**

b. Explain Static and Dynamic hashing in detail. **(08 Marks)**

c. Define the term File Organization. Explain indexed sequential File Organization. **(04 Marks)**

* * * * *

Karnatak Law Society's
Vishwanathrao Deshpande Institute of Technology,
Haliyal 581329
Department of Computer Science & Engg.

Sub: Data Structures & Applications

Subcode: 18CS32

Staff Name: Prof. Yasmeen Shaikh

Question paper: 3rd sem BE Degree Exam, Feb Mar 2022.

Module 1

1a Define Data Structures. Explain the various operations on data structures. [6M]

→ Data may be organized in many different way, the logical or Mathematical model of a particular organization of data is called a data structure.
Ex Stack, Queue, Tree, linked list etc.
The various operations on data structures are

i) Traversing : It is a process of accessing each record exactly once so that certain item can be processed.

ii) Searching : It is process of finding a particular element or all elements in a list.

iii) Inserting : It is a process of adding new record.

iv) Deleting : It is a process of deleting a record.

1b. Define Structures. Explain the types of structures with examples for each. [7M]

→ A Structure is defined as a collection of variables of same data type a dissimilar data type grouped together under a single name.

Syntax

```
struct tagname
{
  datatype member1;
  datatype member2;
  datatype member3;
};
```

example.

```
struct student
{
  char name[25];
  int usn;
  float marks;
};
```

where struct is a keyword which informs the compiler that structure is being defined.

tagname : name of the structure :

member1, member2.. : members of structure.

type1, type2 : ~~in~~ float, int, char, double.

Three ways of declaring structure are
i) Tagged structure
ii) structure without tag
iii) tag defined structure

1. Tagged structure

Syntax

```
struct tag-name
{
  datatype member1;
  datatype member2;
  ...
  ...
};
struct tagname v1, v2, ... vn
```

example.

```
struct student
{
  char name[20];
  int usn;
  float marks;
};
struct student s1, s2, s3;
```

ii) Structure without tagname

Syntax

```
struct
{
 datatype member1 ;
 datatype member2 ;
   ...
   ...
} v1, v2, v3 ;
```

example

```
struct
{
 char name[20];
 int usn ;
 float marks ;
} s1, s2, s3 ;
```

iii) Type defined structure

Syntax

```
typedef struct
{
 datatype member1 ;
 datatype member2 ;
   ...
   ...
} TYPE-ID;
TYPE-ID v1, v2, ... vn ;
```

example:

```
typedef struct
{
 char name[20];
 int usn ;
 float marks ;
} STUDENT;
STUDENT s1, s2, s3 ;
```

1c List & explain the functions supported in c for Dynamic Memory Allocation. [7M]

→ The functions supported in c for dynamic memory allocation are

i) malloc() ,
ii) calloc()
iii) realloc()
iv) free()

i) malloc(size)

- This function allows the program to allocate a block of memory space as & when needed and the exact amount needed during execution.

Syntax: ptr = (data-type) * malloc (size

Where ptr is pointer value of type datatype
data-type is any basic data type
size → no. of bytes.

example.
```
int *ptr
ptr = (int *) malloc (10);
...
...
if (ptr = =NULL)
{
  printf(" Insufficient memory ");
  return;
}
```

The compiler reserves 10 bytes of memory.
If memory cannot be allocated it returns NULL

ii) calloc(n, size)

- calloc stands for contiguous allocation of multiple blocks of memory & is mainly used to allocate memory for arrays.
- The number of blocks is determined by the parameter n.

Syntax: ptr = (data-type *) calloc (n, size);
where ptr is a pointer variable of type datatype
data-type is any basic data type
n is number of blocks to be allocated
size is number of bytes of each block.

example:

```
int *ptr;
ptr = (int *) calloc(5, sizeof(int));
...
...
if (ptr == NULL)
  {
   printf("Insufficient memory");
   return;
  }
```

iii) realloc(ptr, size)

- The realloc() changes the size of the block by extending or deleting the memory at the end of the block.
- If the existing memory can be extended, ptr value will not be changed.
- If the memory cannot be extended, this function allocates a completely new block & copies the contents of the existing memory block into new memory block & then deletes old memory block

```
ptr = (data-type *) realloc (ptr, size);
```

examples

```
main()
{ char *str;
  str = (char *) malloc(10);
  strcpy(str, "COMPUTER");
  str = (char *) realloc(str, 40);
  strcpy(str, "COMPUTER SCIENCE AND ENGG");
}
```

iv) free(ptr)

- This function is used to deallocate the allocated block of memory which is allocated using functions calloc(), malloc(), or realloc();

- free(ptr);
  ptr = NULL;

- It is the responsibility of the programmer to deallocate the memory whenever it is not required by the program & initialize ptr to NULL.

2a. Define pattern matching. Write the Knutt Morris Pratt pattern matching algorithm and apply the same to search the pattern abcdabcy in the text abcxabcdabxabcdabcy. [10M]

→ Given a string called pattern p with m characters & another string called text with n characters where $m \leq n$. It is required to search for the pattern string p in the text string t. If search is successful return the position of the first occurrence of pattern string p in the text string t, otherwise return string p in the text string t, otherwise return -1. This process of searching for a pattern string in a given text string is called pattern matching.

```
int pmatch ( chae *string , chae *pat)
{ int i=0, j=0;
  int lens =strlen (string);
  int lenp =strlen (pat );
   while (i < lens && j < lenp)
     { if (string [i] = = pat [j] )
          {
             i++; j++;
          }
       else if (j = =0)
             i++;
       else  j = failure [j-1 ]+1;
   }
  return ( (j= =lenp) ? (i -lenp) : -1);
}
```

```
a b c x a b c d a b x a b c d a b c d a b c y
a b c d a b c y
  a b c d a b c y
      a b c d a b c y
         a b c d a b c y
            a b c d a b c y
               a b c d a b c y
                  a b c d a b c y
                     a b c d a b c y
                        a b c d a b c y
                           a b c d a b c y
                              a b c d a b c y
                                 a b c d a b c y
                                    a b c d a b c y
                                       a b c d a b c y
                                                (pattern found)
```

```
void fail (char *pat)
{ int   n=strlen(pat);
    failure [0] = -1;
    for (j=1 ; j<n ; j++)
      {
        i = failure [j-1];
        while ((pat [j] != pat [i+1]) && (i>=0))
            i = failure [i];
        if (pat [j] = pat [i+1])
            failure (j) = i+1;
        else failure (j) = -1;
    }
  }
```

2b. Write the Fast Transpose algorithm to transpose the given sparse matrix. Express the given sparse matrix as triplets & find its transpose [10M]

$$A = \begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 10 & 0 & 0 & 25 & 0 \\ 1 & 0 & 23 & 0 & 0 & 45 \\ 2 & 0 & 0 & 0 & 0 & 32 \\ 3 & 42 & 0 & 0 & 31 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 30 & 0 & 0 \end{array}$$

→
```
void fastTranspose(term a[], term b[])
{
  int rowTerms [max-col], startingpos [max-col];
  int i, j, numcols = a[0].col, numTerms = a[0].value;
  b[0].row = numcols;  b[0].col = a[0].row;
  b[0].value = numTerms;
if (numTerms > 0)
{
  for (i=0; i< numcols ; i++)
     rowTerms[i] = 0 ;
  for (i = 1; i< = numTerms ; i++)
     rowTerms [a[i]. col ]++;
  Starting Pos [0] = 1;
  for( i=1; i < numcols ; i++)
     Starting Pos[i] = StartingPos[i-1] + rowTerms[i-1];
  for (i=1; i<=numTerms ; i++)
   {
    j= StartingPos[a[i].col] ++;
    b[j] .row = a[i]. col ;
    b[j] . col = a[i] . row;
    b[j] . value = a[i]. Value ;
   }
 }
}
```

Input of original matrix

	row	col	value
a[0]	6	5	8
a[1]	0	0	10
a[2]	0	3	25
a[3]	1	1	23
a[4]	1	4	45
a[5]	2	4	32
a[6]	3	0	42
a[7]	3	3	31
a[8]	5	2	30

Transpose of given matrix

	row	col	value
b[0]	5	6	8
b[1]	0	0	10
b[2]	0	3	42
b[3]	1	1	23
b[4]	2	5	30
b[5]	3	0	25
b[6]	3	3	31
b[7]	4	1	45
b[8]	4	2	32

Module - 2

3a Define stack. List and explain various operations on stacks using arrays with stack overflow & stack underflow conditions [10M]

→ Stack is an ordered collection of items, into which items may be inserted ~~fro~~ and from which items may be deleted at one end called the top of the stack.

- Stack is also called as LIFO i.e. last in first out data structure because element [illegible] at the last is deleted first.

The various operations on stack are

i) Insertion (push)
ii) Deletion (pop)
iii) Display.

i) <u>Push operation</u> :

To insert element into the stack, we first check whether stack is full, if yes overflow message is displayed & insertion is not possible. If stack is not full, then top is incremented by one & element is inserted.

```
Void push()
{ if (top = = stack-size -1) // stack full condition
  {
    printf (" stack overflow");
    return;
  }
```

```
    top= top+1 ;
    s[top] = item ;
}
```

ii) <u>Pop operation</u>

While deleting element from stack we check whether stack is empty. If it is empty, deletion is not possible & underflow message is displayed.

If stack is not empty, top element from the stack is accessed & deleted & top is decremented by 1.

```
int pop()
{
  int itemdeleted ;
  if (top == -1)   // Stack empty condition
     return 0
  else
    {
     item deleted = s[top--];
    }
    return item deleted ;
}
```

iii) <u>Display operation</u>

```
void display()
{
  int i ;
  if (top == -1)
    {
      printf(" stack is empty \n");
      return;
    }
printf(" contents of stack are \n");
for(i=0; i<=top; i++)
    printf(" %d\n", s[i]);
}
```

The above function display all elements of the stack.

3b Write an algorithm to convert an infix expression to postfix expression & also trace the same for the expression (a+b) * d + e/f + c. [10M]

→ Algorithm

```
infix_to_postfix(infix, postfix)
   opstk = the empty stack
   while (not end of input)
   {
      symb = next input character;
      If (symb is an operand
           add symb to the postfix string
      else
       {
         while( ! empty (opstk) && pred (stacktop
                                  (opstk , symb))
         {
            topsymb = top (opstk);
            add topsymb to the postfix string;
         }
        push (opstk , symb);
      }//end else
   } // end while
```

```
//output any remaining operators */
While ( ! empty (opstk))
{
    topsymb = pop (opstk);
    add topsymb to the postfix string;
} end while
end of algorithm
```

<u>Tracing</u>

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

Stack	S[top]	Symbol	F(S[top]) > G(symbol)	postfix
#	#	(	-1 != 9 , push (	
#(	(	a	0 != 7 push a	
#(a	a	+	8 > 1 pop a	ab+d*ef/+c+
#(	(	+	0 != 1 push +	
#(+	+	b	2 != 7 push b	
#(+b	b	)	8 > 0 pop b	
#(+	+	)	2 > 0 pop +	
# (	(	)	0 = 0 pop (	
#	#	*	-1 != 3 push *	
#*	*	d	4 != 7 push d	
#*d	d	+	8 > 1 pop d	
#*	*	+	4 > 1 pop *	
#	#	+	-1 != 1 push +	
#+	+	e	2 != 7 push e	
#+e	e	/	8 > 3 pop e	
#+	+	/	2 != 3 push /	
#+/	/	f	4 != 7 push f	
#+/f	f	+	8 > 1 pop f	
#+/	/	+	4 > 1 pop /	

Stack	S[top]	Symbol	F[S[top]] > G(symbol)	postfix
# +	+	+	2>1 pop +	
#	#	+	-1 != 1 push +	
#+	+	c	2 != 7 push c	
# + c	end of for loop			
# + c	c != # pop c			
# +	- != # pop +			
#	# = # end of while			

postfix expression is : ab+d*ef/+c+

4a. Define Recursion. Explain the types of recursion. Write the recursive function for
i) factorial of a number ii) Tower of Hanoi [10M]

→ A function which calls itself repeatedly is called recursion.

Types of recursion
i) Direct ~~recursion~~ recursion.
ii) Indirect recursion

i) Direct recursion

A function fun is called direct recursive if i
calls the same function fun.

ex
```
int fact(int n)
{
  if (n==0)
    return 1;
return n* fact(n-1);
}
```

ii) Indirect Recursion

A function fun is called indirect recursive if it calls another function say fun-new & fun-new calls fun directly or indirectly.

example:
```
fun()
{
    ---
    ---
    new-fun();
}

new-fun()
{
    ---
    --
    fun();
}
```

i) Factorial of a number

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

ii) Tower of Hanoi

```
void tower (int n, char A, char C, char B)
{ if (n==1)
  {
    printf(" %s %c %s %c ", "move disk 1",
        "from peg", A, "to peg", C);
        return;
    }
    towers(n-1, A,B,C);
    Printf(" %s %d %s %c %s %c ", "move disk",
          n, "from peg", A, " to peg", C);
    towers (n-1, B,C,A);
}
```

4b Give the Ackermann function & apply the same to evaluate A(1,2). [4M]

→ Ackermann function

```
int A (int m, int n)
{
    if(m==0) return n+1;
    if (n==0) return A(m-1, 1);
    return A(m-1, A(m,n-1));
}
```

Solution

A(1,2) m=1, n=2

A(1,2) = A(0, A(1,1))

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

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

A(0,1) = 1+1 = 2

A(1,0) = 2

A(1,1) = A(0,2)

A(0,2) = 2+1 = 3

$A(1,1) = 3$

$A(1,2) = A(0,3)$

$A(0,3) = 3+1 = 4$

$\boxed{\therefore A(1,2) = 4}$

4 c). Explain the various operations on circular queues using arrays. [6 M]

→ The operation on circular queue are
i) insertion
ii) deletion
iii) Display

i) <u>insertion</u>

To insert an element, we increment rear by rear = (rear+1)% queuesize.

Before inserting we always check queue is full or not using count == queuesize condition.

```
void insert_circular()
{
  if (count == Queuesize)
  {
    ptf(" Queue overflow\n");
    return;
  }
 rear = (rear+1) % Queue_size);
 q[rear] = item;
 count++;
}
```

ii) deletion

While deleting element, we first check whether queue is empty. if not, item is accessed from front and front is increment by (front +1) % Queuesize.

```
int delete-circular()
{
    int item;
    if (count ==0)
        return -1;
    item = q[front];
    front = (front + 1) % Queuesize;
    count - =1;
  return item;
}.
```

iii) display

```
void display()
{   int i, f;
  if (count ==0)
    { printf ("Queue is empty");
     return;
    }
 printf(" contents of circular queue are \n");
 for (i = 1, f = front ; i< = count ;i++)
     {
        print (" %d \n", q[f]);
        f= (f+1) % Queuesize;
     }
}
```

Module 3

5a Give the node structure to create singly linked list of integers & write the functions to perform the following operations..

i) create a list containing three nodes using front insertion

ii) insert a node 40 at end of the list

iii) delete a node whose data is 30

iv) display the list contents.

→ Node structure

```
struct Node
{
    int info;
    struct node *link;
};
typedef struct node *NODE;
```

getnode function

```
NODE getnode()
{
    NODE x;
    x=(NODE) malloc (sizeof(struct node));
    if (x==NULL)
    {
      printf(" out of memory");
      exit(0);
    }
    return x;
}
```

i) insert at front

```
NODE insert_front(int item, NODE first)
{
    NODE temp;
    temp = getnode();
    temp->info = item;
    temp->link = first;
    return temp;
}
```

ii) insert at end of the list

```
NODE insert_rear(int item, NODE first)
{
    NODE temp, cur;
    temp = getnode();
    temp-> info = item;
    temp-> link = NULL;
    if (first = = NULL)
        return temp;
    cur = first;
    while(cur -> link != NULL)
            cur = cur->link;
    cur->link = temp;
    return first
```

iii) ~~N~~NODE delete_info(int key, NODE first)

```
{
    NODE prev, cur;
    if (first==NULL)
     {
      printf(" list is empty");
      return NULL;
     }
    if(key= first->info)
    { cur = first; ~~fir~~
      first= first ->link;
      free(cur);
```

```
prev=NULL, cur=first
While (cur != NULL)
 { if (key= =cur->info) break
     prev=cur; cur=cur->link;
 }
 if(cur = = NULL)
  { pf(" search is unsuccessfull");
     return;
  }
Prev->link = cur->link
 free(cur)
     return first
```

```
iv) void display(NODE first)
    {
     NODE cur;
     if( first = =NULL)
       {
        printf(" List is empty");
        return;
       }
     printf(" The contents of singly linked list \n");
     cur = first;
     while (cur ! = NULL)
       {
        printf(" %d", cur->info);
        cur = cur->link;
       }
       printf("\n");
    }
```

5b Write the functions for i) finding Length of the list
ii) concatenate two lists iii) Reverse a list [10M]

```
→ i) int length(NODE first)
    {
     NODE cur;
      int count =0
     if (first= =NULL)
        return 0;
     cur = first;
     while (cur ! =NULL)
      {
        count ++;
        cur =cur->link;
      }
     return count;
    }
```

```
ii) NODE Concat(NODE first, NODE sec)
    {
     NODE cur;
     if (first = =NULL) return sec;
     if (sec = =NULL) return first;
     cur=first;
     while (cur->link ! =NULL)
           cur=cur->link;
     cur->link =sec;
     return first;
    }
```

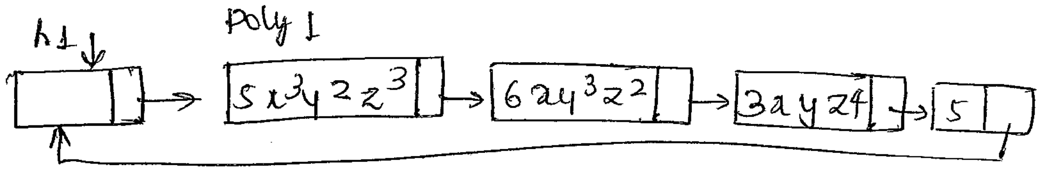
```
iii) NODE reverse (NODE first)
  {
    NODE cur, temp;
    cur = NULL;
    While (first != NULL)
      {
        temp = first -> link;
        first -> link = cur;
        cur = first;
        first = temp;
      }
    return cur;
  }
```

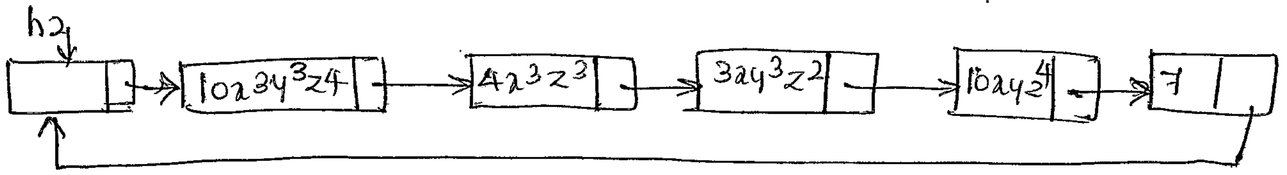
6a Write the node representation for the linked representation of a polynomial. Explain the algorithm to add two polynomials represented as linked list.

→ node representation of: a polynomial

```
Struct node
{
   int cf, px, py, pz;
   Struct node *link;
};
typedef struct node * NODE;
```

Consider two polynomials





If we want to add two polynomials, first we have to search for power of polynomial 1 in polynomial 2.

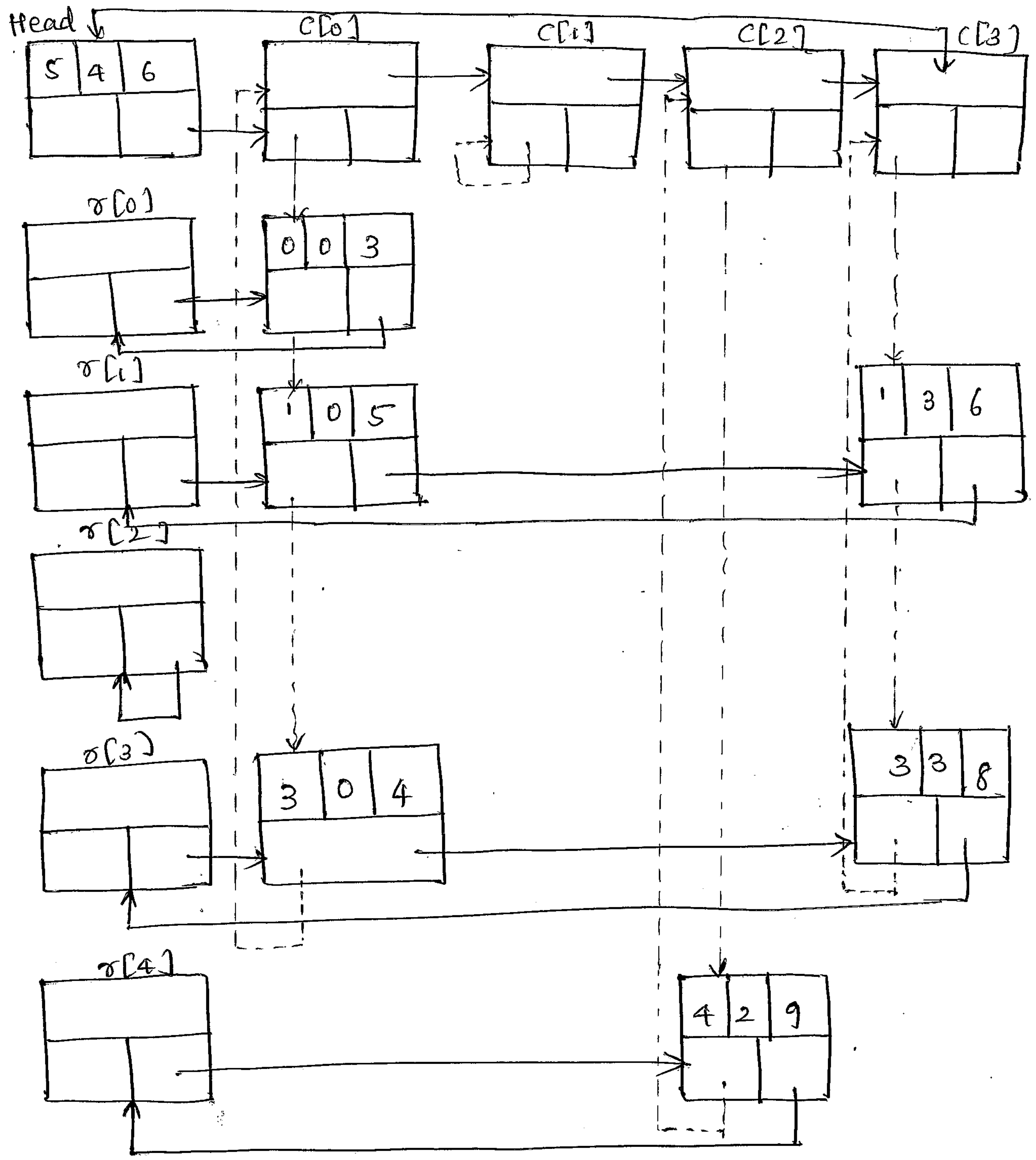
```
NODE search( NODE p1, NODE p2)
{
  int cf1, px1, py1, pz1, cf2, px2, py2, pz2;
  NODE p2;
  cf1 = p1->cf ; px1 = p1->px ; py1 = p1->py ; pz1=p1->pz;
  p2 = h2->link;
  while(p2 != h2)
  {
    cf2=p2->cf; px2=p2->px ; py2 =p2->py ; pz2 =p2->pz;
    if (px1 = = px2 && py1 = =py2 && pz1 = =pz2 ) break;
    p2 = p2->link;
  }
  return p2;
}
```

Once we know to search for a term of polynomial 1 in polynomial 2, the general procedure to add two polynomials is shown below:

```
for each term of polynomial 1
    Step 1 : access each term of poly 1
    Step2 : search for power of above term in poly 2
    Step3 : if found in poly2
                Add the coefficients & add sum to poly 3
            else
                Add the term of poly1 to poly 3
end for
Add remaining terms of poly2 to poly 3.
```

6b For the given sparse matrix, write the diagrammatic linked list representation. [4m]

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 5 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 8 \\ 0 & 0 & 9 & 0 \end{bmatrix}$$

→ 

6c List out the differences between single linked list and double linked list. Write the functions to perform the following operations on double linked list
i) insert a node at rear end of the list
ii) delete a node at rear end of the list
iii) search a node with a given key value [8 M]

→

Singly linked List	Doubly linked list
i) consists of single link field	ii) consists of two links, left & right

ii) Supports traversal in forward direction only	ii) Supports traversal in forward & backward direction
iii) insertion & deletion at rear ~~end~~. is time consuming, because we need to traverse the entire list to reach the last node	iii) address of last node is easily accessible hence insertion & deletion at rear end are faster.

i) insert node at rear end

```
NODE insertrear (int item,
                 NODE first)
{ NODE temp, cur;
  temp = getnode();
  temp->info = item;
  temp->llink = temp->rlink
                    = NULL;
  if (first == NULL) return temp
  cur = first;
  while (cur->rlink != NULL)
      cur = cur->rlink;
  cur->rlink = temp;
  temp->llink = cur;
  return first
}
```

ii) delete node at rear end

```
NODE delete_rear (NODE first)
{
  NODE cur, prev;
  if (first == NULL)
  {
    printf("List is Empty");
    return first;
  }
  if (first->link == NULL)
  {
    printf("item deleted = %d",
                first->info);
    free(first);
    return NULL;
  }
cur = first;
prev = NULL;
while (cur->rlink != NULL)
  { prev = cur;
    cur = cur->rlink;
  }
printf("Item deleted = %d",
                 cur->info);
  free(cur);
  prev->rlink = NULL;
}  return first;
```

iii) search a node with a given key value

```
NODE search (int item, NODE head)
{
    NODE prev, cur, next;
    if(head -> rlink = = head)
     {
      printf(" List is empty");
      return head;
     }
     cur = head -> rlink;
     while (cur != head && item != cur -> info)
          cur = cur -> rlink;
     if(cur = = head)
      {
       printf("Item not found");
       return head;
      }
   printf(" Item found;
   return head;
}
```

Module 4

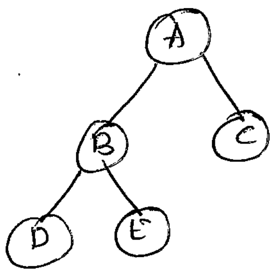
7a. Define Tree. With suitable example explain
i) Binary tree ii) Complete binary tree [10M]
iii) strict binary tree iv) skewed binary tree

→ Tree -

A tree is a set of one or more nodes that show parent - child relation such that
i) There is a special node called root node.

ii) The remaining nodes are partitioned into disjoint subsets T1, T2, ... Tn, n>,0 where T1, T2, ... Tn which are all children of root node are themselves trees called subtrees.

example



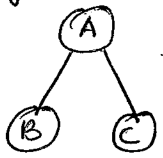
i) Binary tree -

- A Binary tree is a tree which has finite set of nodes that is either empty or consist of a root & two subtrees called left subtree & right subtree.

- Root - If tree is not empty, the first node in the tree is called root node.

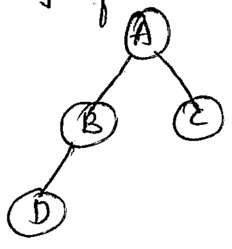
Left subtree - It is a tree which is connected to the left of the root

right subtree - It is a tree which is connected to the right of the root



ii) Complete Binary tree

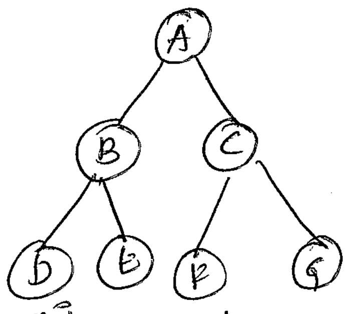
A Complete binary tree is a binary tree in which every level, except possibly the last level is completely filled



iii) Strictly binary tree

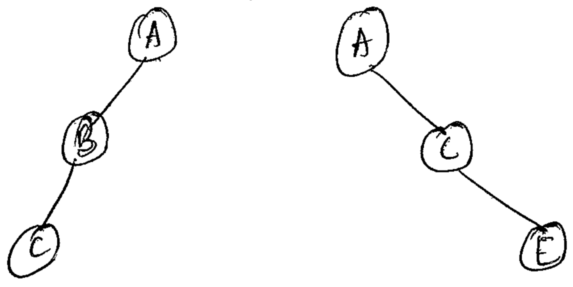
A binary tree having 2^i nodes in any given level i is called strictly binary tree.

->



iv) Skewed Binary tree

A skewed tree is a tree consisting of only left subtree or only right subtree. A tree with only left subtree is called left skewed Binary tree. & a tree with right subtree is called right skewed binary tree.



7b. Write the routines to traverse the given tree using
i) Pre-order traversal   ii) Post-order traversal [6M]

-> i) Pre-order traversal.

```
void preorder (NODE root)
{
  if(root==NULL) return;
  printf ("%d", root->info);
  Preorder (root->llink)
  Preorder (root->rlink)
}
```

ii) Post-order traversal

```
void postorder (NODE root)
{
 if (root ==NULL) return
  postorder (root->llink);
  postorder (root ->rlink);
  Printf(" %d", root->info);
}
```

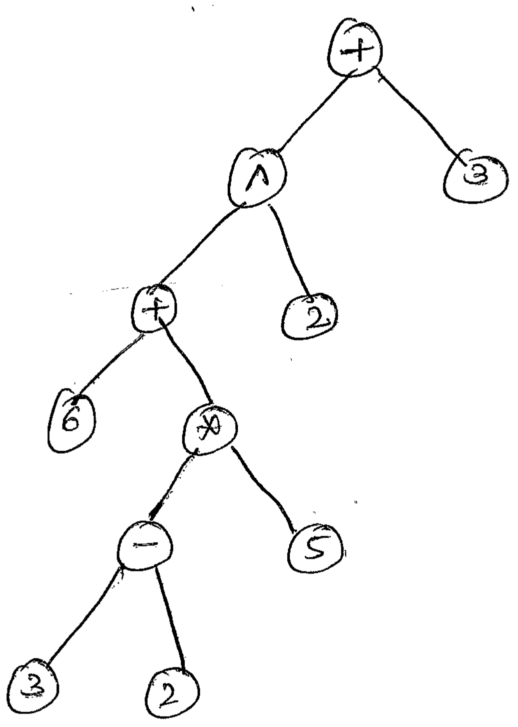
7c Write the recursive search algorithm for a binary search tree. [4M]

→
```
NODE search( int item, NODE root )
{
    if (root == NULL)
        return root;
    if (item == root->info)
        return root;
    if (item < root->info)
        return search(item, root->llink);
    return search (item, root->rlink);
}
```

8a Draw a Binary tree for the following expression ((6+(3-2)*5) ∧ 2+3. Traverse the above generated tree using pre-order & post-order & also write their respective functions. [10M]

→ Consider the given expression:

((6+(3-2)*5) ∧ 2+3.



Post order sequence - $32-5*+2$3+

Preorder sequence - +$+6*-32523

```
void preorder(NODE root)
{
 if(root==NULL) return;
 printf("%d", root->info);
 Preorder (root->llink);
 preorder (root->rlink);
}
```

```
void postorder (NODE root)
{
if (root==NULL) return
postorder (root->llink);
postorder (root->rlink);
printf ("%d", root->info);
}.
```

8b Write the routines for [10M]

i) copying of binary trees ii) Testing equality of binary trees

i) <u>copying of binary trees</u>

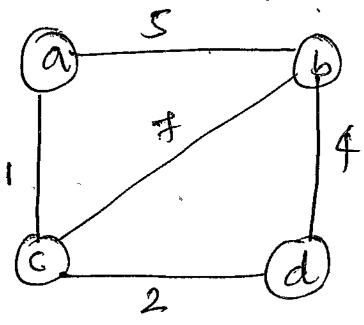
```
NODE copy (NODE root)
{
  NODE temp;
  if (root == NULL)
       return NULL;
  temp = getnode();
  temp->info = root->info;
  temp->lptr = Copy (root->lptr);
  temp->rptr = Copy (root->rptr);
  return temp;
}
```

ii) <u>Testing equality of B.T</u>

```
int equal (NODE r1, NODE r2)
{
 if (r1==NULL && r2==NULL)
     return 1;
 if (r1==NULL && r2!=NULL)
     return 0;
 if (r1!=NULL && r2==NULL)
     return 0;
 if (r1->info != r2->info)
     return 0;
 if (r1->info == r2->info)
     return 1;
 return equal (r1->llink, r2->llink,
     && equal (r1->rlink, r2->rlink)
}
```

Module 5

9a. Define Graphs. Give the Adjacency matrix and Adjacency list representation for the following graph in Fig. Q9(a). [8M]

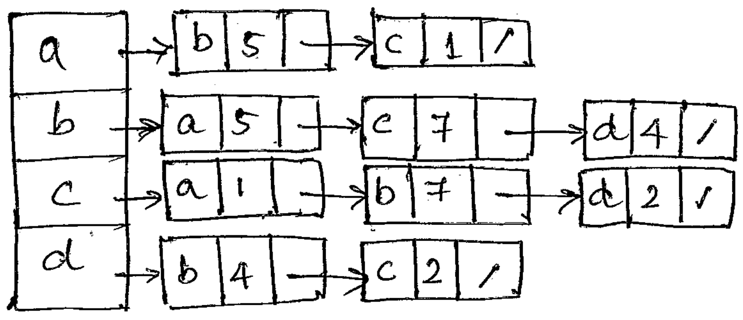


→ A Graph G is defined as pair of two sets V and E denoted by G=(V,E) where V is the set of vertices and E is set of edges.

For the given graph, Adjacency matrix representation is

$$\begin{array}{c|cccc|} & a & b & c & d \\ \hline a & 0 & 5 & 1 & \infty \\ b & 5 & 0 & 7 & 4 \\ c & 1 & 7 & 0 & 2 \\ d & \infty & 4 & 2 & 0 \end{array}$$

For the given graph, Adjacency linked list representation is



9b Write the algorithm for following Graph Traversal methods. [8M]

i) Breadth First search ii) Depth First search

→ i) Breadth first search algorithm

```
BFS(a,n,u)
Begin
```

```
no node is visited to start with
insert source u to q
print u
Mark u as visited i.e. add u to s
while queue is not empty
        delete a vertex u from q
        for every v adjacent to u
        If v is not visited
            print v
            mark v as visited
            insert v to queue
      end if
  end while
end algorithm
```

ii) Depth First search algorithm

Step 1: Select node u as the start vertex (select in alphabetical order), push u onto stack & mark it as visited. we add u to s for marking.

```
Step 2 : while stack is not empty
            For vertex u on top of the stack, find the
            next immediate adjacent vertex.
          If v is adjacent
              If a vertex v. not visited then
                  push it on to stack & number it in
                  the order it is pushed.
                  mark it as visited by adding v to s.
        else
              ignore the vertex
        end if
```

```
else
    remove the vertex from the stack
    number it in the order it is popped.
end if.
end while
```

Step3: Repeat step 1 & step 2 until all the vertices in the graph are considered.

9c Write an algorithm for insertion sort. [4M]

→
```
insertion-sort (a, n)
Begin
    for i = 1 to n do
        item ← a[i]
        j ← i-1
        while item < a[j] && j >= 0.
            a[j+1] ← a[j]
            j ← j-1
        end while
        a[j+1] ← item
    end for
end algorithm
```

10a. Define Hashing. Explain any three Hash functions. [8M]

→ The process of mapping large amount of data into a smaller table using hash function, hash value & hashtable is called hashing.

There are four types of hash-function

i) Division method
ii) Mid square method
iii) Folding method
iv) Converting keys to integers

i) <u>Division method</u>

We choose a number m which is prime value and it is larger than the number of given elements in array a.

Here the key item k is divided by some number m & the remainder is used as the hash value

$h(k) = k \% m$

ex consider elements 11, 12, 13, 14, 15. m=5

$H(11) = 11 \% 5 = 1$

$H(12) = 12 \% 5 = 2$

$H(13) = 13 \% 5 = 3$

$H(14) = 14 \% 5 = 4$

$H(15) = 15 \% 5 = 0$.

ii) <u>Mid square method</u>

In this method, ~~the key k is squared~~. A number in the middle of k^2 is selected by removing the digits from both ends.

The hash function is defined as shown below

$h(k) = l$.

consider key k = 2345

$k^2 = 574525$

$h(2345) = 45$, by removing 57 in the begining & 25 from the end in k^2.

iii) Folding method

In this method, the key k is divided into number of parts k1, k2, ...kn of same length except the last part.

Then all parts are added together as shown below:

$h(k) = k_1 + k_2 + \dots + k_n$

eg consider $K = 123987234876$

$\therefore k_1 = 12, k_2 = 39, k_3 = 87, k_4 = 23, k_5 = 48$

$k_6 = 76$.

hash value $h(k) = k_1 + k_2 + k_3 + k_4 + k_5 + k_6$

$= 12 + 39 + 87 + 23 + 48 + 76$

$= 285$

10b) Explain static hashing and dynamic hashing in detail. [8M]

→ 1) Static hashing

In this type of hashing, the address of the resultant data bucket always will be the same.

Lets take an example, if we want to generate the address for product-ID = 76 using mode (5) hash function, it always provides the result in the same bucket address as 3.

In memory, the number of data buckets always remains the same or constant.

operations of static hashing

1) search a record
2) insert a record
3) delete a record
4) update a record.

i) search a record

When we need to search a record, then the same hash function helps us retrieve the bucket's address where the data is stored.

ii) Insert a record

When we need to insert the new record in the table, the hash function automatically generated the address bucket for the record based on the hash key.

iii) Delete a record

When we want to delete the record from the table, we will read or fetch the record, which we want to be deleted using the hash function. And, after that, we have to remove the record for that memory address.

iv) update a record

When we want to update the existing record in the table, then by using the hash function, firstly we have to search the record which we wish to update. Then the data record is updated automatically.

Following are the two different methods which overcome the problem of bucket overflow situation.

i) open hashing ii) closed hashing

i) open hashing

In this method of hashing, the next free data block is selected for entering the new record, rather than overwriting the data to the previous block. This mechanism or method is also known as linear probing.

ii) closed hashing

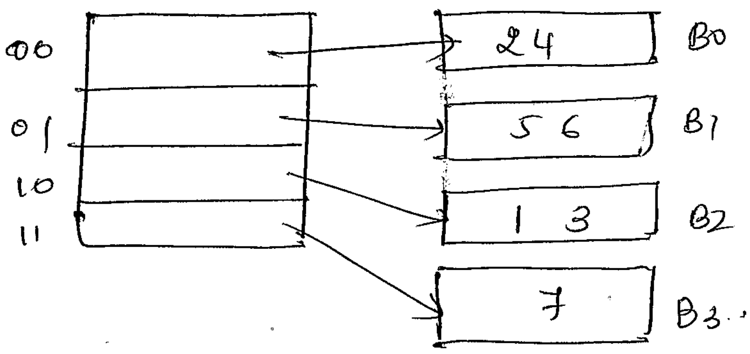
In this method of hashing, when the data bucket is filled with data, then the new bucket is selected for the same hash result & is linked with the previously filled bucket. This mechanism is called overflow-chaining.

2) Dynamic hashing

This type of hashing is introduced to solve the basic problem of static hashing. The problem of static hashing technique is that it does not reduce or enlarge its size dynamically according to the database size.

In this type of hashing, the data buckets are dynamically added or removed as the records in the database increase or decrease.

- This type of hashing is called called extended hashing. The hash function is created for generating a large number of values



10c Define the term file organization. Explain indexed file organization [4M].

→ The method of storing files in memory in certain order so that the insertion or deletion or modification will be faster & efficient thereby increasing the performance of the system is called file organization

i) Indexed file organization

- An Indexed sequential file uses the concept of both sequential as well as relative files.
- The indexed sequential files are stored in the disk in order in which they were written to the disk.
- All records can be retrieved in sequential order or in random order using a numeric index.

The file organization stores data for fast retrieval

- The records in an indexed sequential file are of fixed length & every record is uniquely identified by a key field.

- ex To read all records from an indexed sequential file in order, we have to open the file to read the records without specifying an index.
- Thus, sequentially each record is read from the file, till we encounter end of file.
- By specifying the key, we can access the record directly from the disk

Advantages

- Since indices are small, records can be searched very quickly.
- Records can be accessed sequentially & randomly.

Disadvantages

- They can be stored only a disks.
- It requires extra storage space for storing indices.
- Supports only fixed length records.

[illegible]
Faculty Name & Signature
Prof. Yasmeen . Shaikh

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

[illegible]
Dean Academics
Dean, Academics
KLS VDIT, HALIYAL