# B. TECH. TCS-302

## THIRD SEMESTER EXAMINATION, 2008-09

# DATA STRUCTURE USING 'C'

*Time :* 3 *Hours* *Total Marks :* 100

**1. Attempt any two parts of the following:** $10 \times 2 = 20$

**(a) (i) Define the term Data Structure. List some linear and non-linear data structures stating the application area where they will be used.**

**Ans.** Data may be organized in many different ways; the logical or mathematical model of a particular organization of data is called a data structure.

Data structures are classified as either linear or non-linear. A data structure is said to be linear if its elements form a sequence, or in other words, a linear list. There are two basic ways of representing such linear structures in the memory. One way is to have the linear relationship between the elements represented by means of sequential memory locations. These linear structure are called arrays. The other ways is to have the linear relationship between the elements represented by means of pointer or links. The linear structures are called linked lists.

Ex: String, arrays, lists, stacks and queues.

Tree is a non-linear data structure which is mainly used to represent data containing a hierarchical between elements.

Ex: records, family trees and tables of contents, graph.

**(ii) State the merits and demerits of static and dynamic memory allocation techniques.**

**Ans. MERITS:**

**Static:**

(1) In static memory allocation method memory is allocated of statically so there is only fixed amount of memory needed.

(2) It requires less over heads of data structure.

(3) It also requires less memory management.

(4) It uses Array type data structure.

**DYNAMIC:**

(1) It provides a flexible data structure.

(2) In dynamic memory allocation method memory can be allocated according to need.

(3) It uses calloc () and malloc () functions.

**DEMERITS:**

**Static**

- In static memory allocation since size of memory once allocated cannot expand or shrink.

**DYNAMIC**

- In dynamic memory allocation method their is extra over heads of memory management.
- In this method we uses different type of garbage collection technique uses for

collecting free spaces. It needs complex memory management techniques.

**(b) (i) Define tail recursions by giving suitable example.**

**Ans.** The process in which function calls itself in function body is called as recursion. But when this called function is the last executed statement in the function body then it is called tail recursion.

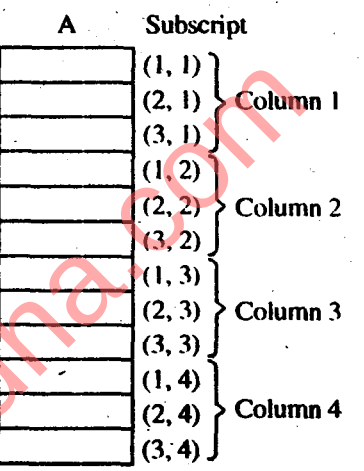
Here it will not append new function in stack, but it will overwrite the value of previous function with the current one. So it will give better performance.

Ex: Factorial:

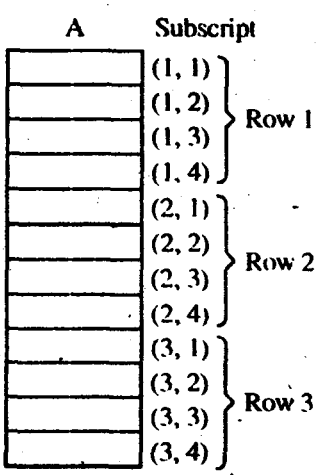
```
≠include < stdio . h>
void mainc )
  {
    int n, value;
    printf ("enter the no.:");
    scanf ("% d", and n);
    if (n < 0)
       printf ("No factorial of -ve no");
    else
       if (n = = 0)
          printf ("Factorial of zero is 1/ n");
    else
    {
       valuc = factorial (n, 1);
       printf ("Factorial of % d = / n", n, value);
    }
    factorial (int n, int fact)
    {
       if (n = = 1)
       return fact;
    else
       factorial (n-1, n * fact);
    }
```

**(ii) How two dimensional arrays are represented in memory? Also obtain the formula for calculating the address of any element stored in the array, in case of column major order. (Make necessary assumptions yourself).**

**Ans.** Let A be a two dimensional m × n array. Although A is pictured as a rectangular array of elements with m rows and n columns, the array reprinted in memory by a block of mn sequential memory locations.




(*a*) Column-major order




(*b*) Row-major order

The computer keeps track of Base (A)—the address of the first element A[1, 1] of – and compute the address LOC (A[J, K]) of A[J, K] using the formula

LOC (A[J,K]) = Base (A) + w[M[k – 1] + (J – 1)]

in column major order where w is the no. of bit in per block.

LOC (A[J, K]) = Base (A) + w [N(J – 1) + (k – 1)].

**(c) Write an algorithm to convert a valid arithmetic infix expression into its equivalent positive expression.**

**Trace your algorithm for**

**A–B/C+D*E+F**

**Ans.** POLISH (Q, P)

(1) Push "("onto STACK, and add")" to end of Q.

(2) Scan Q from left to right and respect steps 3 to 6 for each element of Q until the STACK is empty.

(3) If an operand is encountered, add it to P.

(4) If a left parenthesis is ,, , push it onto STACK.

(5) If an operator (×) is ,, , then:

(a) Repeatedly pop from STACK and add to P each operator which has the same precedence as or higher precedence than (×).

(b) Add (×) to STACK.

(6) If a right parentheses is encountered, then

(a) repeatedly pop from STACK and add to P each operator until a left parenthesis.

(b) Remove the left parenthesis.

(7) Exit.

A–B/C+D*E+F

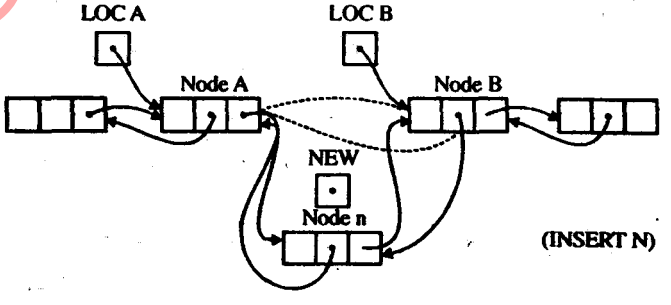
| Symbol-Scanned | STACK | Expression P |
|---|---|---|
| A | ( | A |
| – | (– | A |
| B | (– | AB |
| / | (–/ | AB |
| C | (–/ | ABC |
| + | (/+ | ABC / – |
| D | (+ | ABC / – D |
| * | (+ * | ABC / – D |
| E | (+ * | ABC / – DE |
| + | (+ | ABC / – DE * + |
| F | (+ | ABC / – DE * + F |
| ) | | ABC / – DE * + F + |

**2. Attempt any two parts of the following:** **10 × 2 = 20**

**(a) Write an algorithm to insert and delete a node from doubly linked list. Illustrate with an example. Use your algorithm to develop functions in C.**

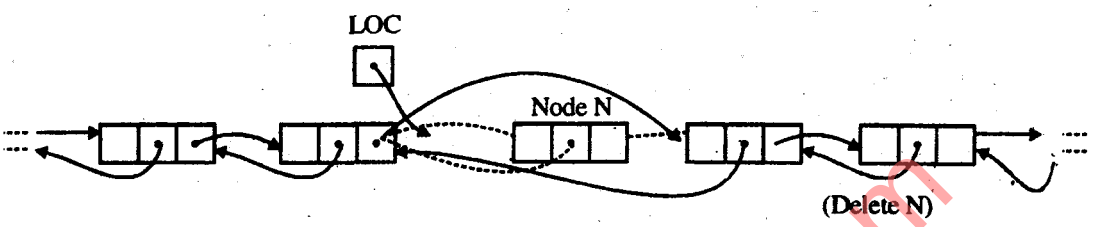
**Ans.** INSDLL (INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

1. IF AVAIL = NULL, then write: OVERFLOW, AND Exit.
2. NEW = AVAIL, AVAIL = FORW [AVAIL], INFO [NEW] : = IT EM
3. FORW [LOCA] : NEW, FORW [NEW] : = LOCB,
4. Exit



DELDLL (INFO, FORW, BACK, START, AVAIL, LOC)

1. FORW [BACK [LOC]] : = FORW [LOC] and
   BACK [FORW [LOC]] : = BACK [LOC]
2. FORW [LOC] : = AVAIL and AVAIL : = LOC
3. Exit.

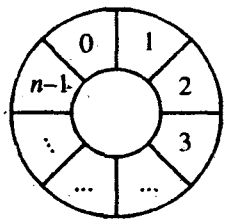



```
Program :-
# include < stdio.h>
# include < malloc.h>
struct node
  {
    struct node * prev;
    int info;
    struct node * next;
  }*start;
void main ()
  {
    int choice, m,n;
    start = NULL
    while (1)
      {
        printf ("1 insertion/n");
        printf ("2.deletion/n");
        printf ("3 . display/n");
        printf ("4 . exit / n");
        printf ("Enter your choice)
    Switch (choice)
    {
Case 1:
  printf ("enter element:");
    scanf ("% d", & m);
    Insert (m);
Case 2;
    printf ("deleted element :");
    scanf ("% d", & m);
    del (m);
    break
Case 3:
    display ();
    break;
Case 4:
    exit);
      }
    }
}
Insert (int num)
{
    struct node * tmp;
    tmp = malloc (size of (struct node));
    tmp → prev = NULL;
    tmp → info = num;
    tmp → next = start;
    start → prev = tmp;
    start → tmp;
```

```
}
del (int num)
{
   struct node * Tmp, * q; q = start;
while (q → next→ next ! NULL)
   {
      if (q → next → info = = num)
         {
            tmp = g → next;
            g → next = tmp→ next;
            tmp → next → prev = g;
            free (tmp);
            return;
         }
         q = q → next;
      }
display ()
   {
      struct node * q;
      if (Start = = NULL)
   {
      printf ("List i, empty /n");
      return;
   }
   q = start;
   printf ("List i, : /n");
   white (q ! = NULL)
   {
      printf ("% d", q → info);
      q = q → next;
   }
   printf ("in");
   }
```

**(b) Explain circular queue and priority queue data structure in detail.**

**Ans. Circular queue:** As in a cirde after last element, first element occurs.




Here after $(n-1)^{th}$ element, $O^{th}$ element occurs.

**Insertion:**

```
If (front = = 0 && rear = = MAXSIZE – 1) ll (fornt
= = rear + 1))
{
   printf ("Queue over flow /n");
   return;
}
   if (front = = –1)
   {
   front = 0;
   rear = 0;
}
   else
   if (rear = MAXSIZE – 1)
      rear = 0;
   else
      rear = rear +1;
      equere - over [rear] = item;
```

**Deletion:**

```
   if (front = = MAXSIZE –1)
      front = 0;
   if (front = = rear)
   {
      front = –1;
      rear = –1;
   }
```


$f = 1$ $r = 3$

(i) initial


$f = 2$ $r = 3$

(ii) deletion

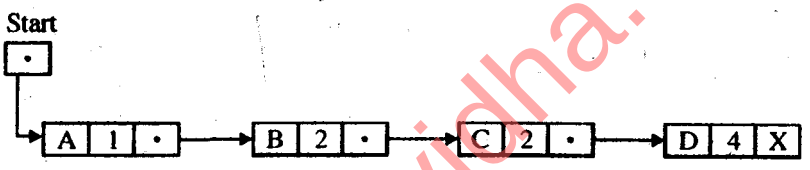

$r = 3$ $f = 2$

(ii) insertion

A priority queue is a collection of elements s.t. each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules:

(1) An element of higher priority 1, processed before any element of lower priority.

(2) Two element with same priority are processed according to the order in which they were added to the queue.




**Deletion:**

(1) ITEM: = INFO [START].

(2) Delete first node from the list

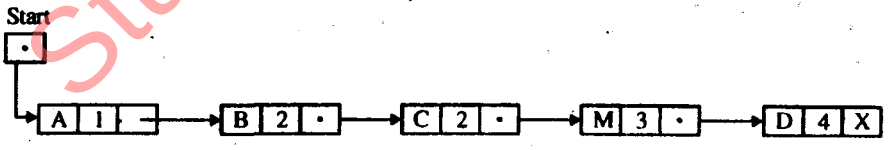
(3) Process ITEM

(4) Exist

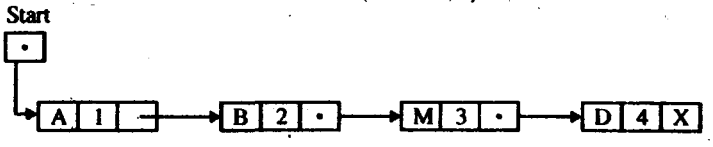
**Insertion:**

(a) Traverse the one-way list until finding a node X where priority no exceeds N. Insert ITEM is front of node X.

(b) If no such node is found, insert ITEM as the last element of the list




(insert M, 3)



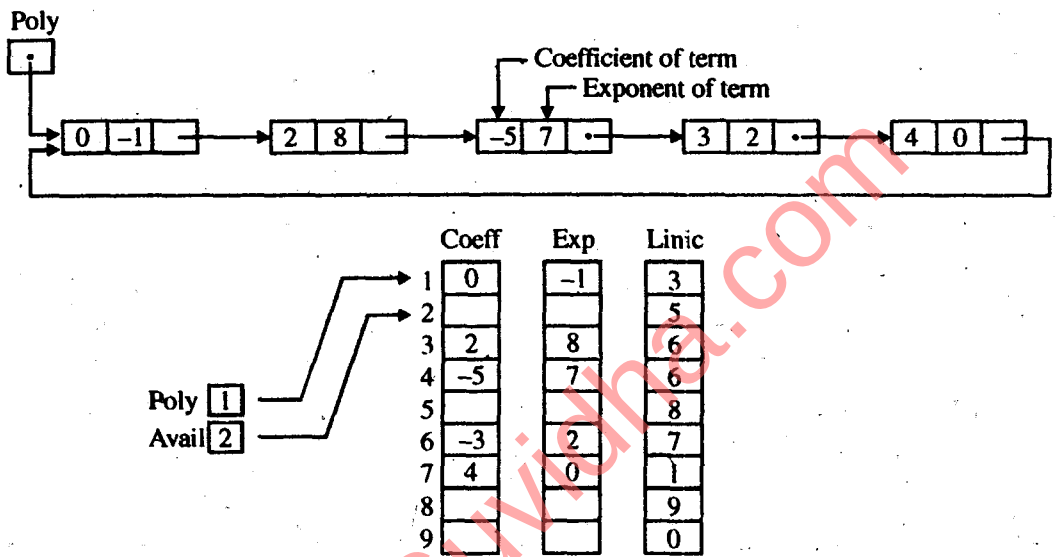

(delete C)

**(c) (i) How can we represent a polynomial in a linked list? Write an algorithm to add two polynomials represented by linked lists.**

**(ii) Explain the term Garbage collection and compaction.**

**Ans.** (i) Let $p(x)$ denote the following polynomial in one variable

$$p(x) = 2x^8 - 5x^7 - 3x^2 + 4$$




POLY - ADD (P1, P2, P)

1. if P1 = NULL && P2 = NULL
   then return NULL
2. while (P1 ≠ NULL && P2 ≠ NULL)
   do

if (exp[P1] > exp[P2]
then exp[P] = exp[P1]
P1 = LINK [P1]
else if (exp[P2] > exp[P1])
then
exp[P] = exp[P2]
coeff[P] = coeff[P2]
P2 = LINK [P2]
else
if (exp[P1] = exp[P2])
then
coeff[P] = coeff[P1] + coeff[P2]
exp[P] = exp[P1]
P1 = LINK [P1]
P2 = LINK [P2]2
do
coeff[P1] = coeff[P1]
exp[1°] = exp [P1]
P1 = LINK[P1]
while P2 ≠ NULL
do
coeff[P] = coeff[P2]
exp[P] = exp[P2]
P2 = LINIK [P2]

**(ii) Garbage collection and compaction**

**Ans.** The operating system of a computer may periodically collect all the deleted space onto the free storage list. Any technique which does this collection i, called "garbage collection". Garbage collection usually takes place in two steps.

First the computer runs through all lists, tagging those cells which are currently in use, and then the computer runs through the memory, collection all untagged space onto the free storage list.

The garbage collection may take place where there is only some maximum amount of space or no space at all left in the free -stage list, or when the CPU is idle and has time to do the collection.

Generally speaking, the garbage collection is invisible to the programmer.

**3. Attempt any four parts of the following:**

**(a) Write an algorithm to insert a node in a threaded binary tree.**

**Ans.** INSTBT (INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM, LOC)

1. Call FIND (INFO, LEFT, RIGHT, ROOT, ITENA, LOC, PAR
2. If LOC ≠ NULL, then exist
3. (a) If AVAIL = NULL, then
   write : OVERFLOW & Exit
   (b) NEW = AVAIL, AVAIL = LEFT [AVAIL]
   and INFO [N ∈ W] = ITEM
   (c) LOC = N ∈ W
   LEFT [N ∈ W] = NULL
   and RIGHT [N ∈ W] = NULL
4. If PAR = NULL, then
   ROOT = N ∈ W
   Else if ITEM < INFO [PAR], then
   LEFT [PAR] = N ∈ W
   Else
   RIGHT [PAR] = N ∈ W
5. Exit

**(b) Discuss how can handle overflow in hashing.**

**Ans.** There are chances of insertion failure when hash table is full. So the solution for this particular case is to create a new hash table with the double size of previous hash table. Here we will use new hash function and we will insert all the elements of the previous hash table. So we will scan the element, of " " " one by one and calculate the hash key with new hash function and insert then into new hash table. This technique is called rehashing

| | |
|---|---|
| 0 | 10 |
| 1 | |
| 2 | 46 |
| 3 | 36 |
| 4 | 25 |
| 5 | |
| 6 | |
| 7 | 7 |
| 8 | 18 |
| 9 | |
| 10 | 43 |

| | |
|---|---|
| 0 | 46 |
| 1 | |
| 2 | 25 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | 7 |
| 8 | |
| 9 | |
| 10 | 10 |
| 11 | |
| 12 | |
| 13 | 36 |

| | |
|---|---|
| 14 | |
| 15 | |
| 16 | |
| 17 | |
| 18 | 18 |
| 19 | |
| 20 | 43 |
| 21 | |
| 22 | |

**(c) Define Tree. How a tree can be stored in memory? Explain with an example.**

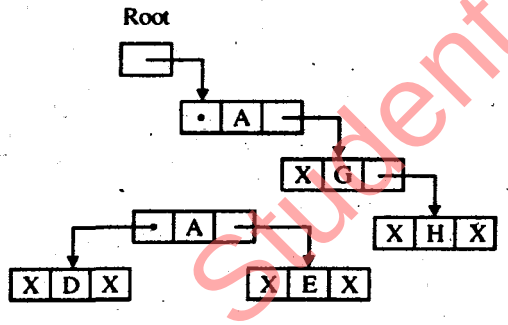
**Ans.** A tree is a nonlinear data containing a hierarchical relationship between elements.

A tree is a graph without any circuit. A binary tree T is defined as a finite sets of elements, called nodes.

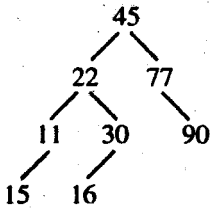
(a) T is empty or

(b) T containing a distinguished node R, called the root of T, and the remaining nodes of T *form an ordered pair of disjoint binary tree* $T_1$ and $T_2$.

(i) Linked representation in memory:




(ii) Sequential representation in memory:

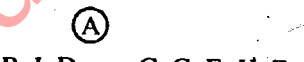


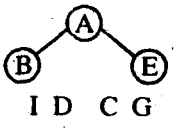

| | |
|---|---|
| 1 | 45 |
| 2 | 22 |
| 3 | 77 |
| 4 | 11 |
| 5 | 30 |
| 6 | |
| 7 | 90 |
| 8 | |
| 9 | 15 |
| 10 | 16 |
| 11 | |
| 12 | |
| . | |

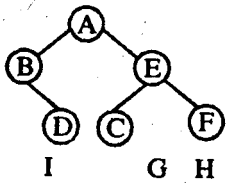
**(e) If the inorder traversal of a binary tree is B, I, D, A, C, G, E, H, F and its post-order traversal is I, D, B, G, C, H, F, E, A, determine the binary tree.**

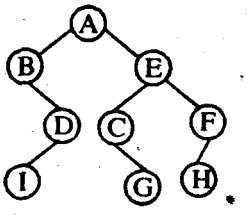
**Ans.** In order: B I D A C G E H F

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

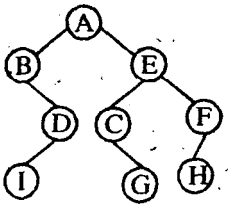


**Ans.**




**(f) Write a program in C for binary search. What is the worst case time complexity of binary search?**

*Ans.*

```
≠ include < stdio.h>
void main ()
{
  int arr[20], start, end, middle, n, i, item;
  printf ("How many elements you want to enter
  in the array";
  scanf ("%d", &n)
  for (i = 0; i < n; i ++)
  {
    printf ("enter element %d: ", i+1);
    scanf ("%d", & arr [i]);
  }
  printf ("enter the element to be searched :");
  scanf ("%d", & item);
  start = 0;
  end = n–1;
  middle = (start + end) /2;
  while (item ! = arr [middle] && start < = end)
  {
    if (item > arr [middle])
      start = middle + 1;
    else
      end = middle–1;
    middle = (start + end) /2;
  }
  if (item = = arr [middle])
    printf ("%d found at posn %d /n", item,
      middle +1);
  if (start > end)
    printf ("%d not found in array /n", item);
  getch ();
  }
```

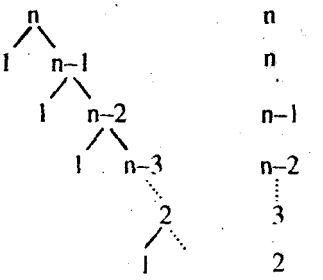
Time complexity —the worst case time complexity is o ($\log_2 n$).

**4. Attempt any four parts of the following:** **5 × 4 = 20**

**(a) How the choice of pivot element affect the running time of quick sort algorithm? Explain.**

**Ans.** (i) **If the pivot element is either smallest or longest element:**

In this case the worst case running time occurs. In this case the one part empty (i) & other contain the remaining (n–1) elements



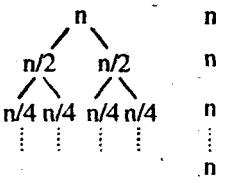

$T(n) = T(n-1) + (n) = \Theta(n^2)$

(ii) **If the pivot element in middle :** If the partitioning procedure produces two regions of size n/2, quick sort, run much faster: so then

$T(n) = 2T(n/2) + \Theta(n)$

$T(n) = \Theta(n \log n)$

(Best case)

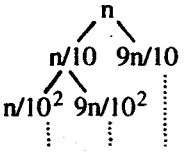


(iii) **Randomly chosen:**

Average case occur.

$T(n) = T(9n/10) + T(n/10) + \Theta(n)$

$= \Theta(n \log n)$




**(b) Write Heapsort algorithm. Analyze the running item of your algorithm.**

**Ans.** HEAPSORT (A, N)

1. [Build a Heap H]
   Repeat for J = 1 to N – 1;
   CALL INSHEAP (A, J, A (J + 1]).
   [END of LOOP]
2. [Sort A by repeatedly deleting the root of H]
   Repeat while N > 1:
   (a) CALL DELHEAP (A, N, ITEM)
   (b) A [N + 1]: = ITEM
   [END of LOOP]
3. Exit.

**Running Time:** The running time of leapsort is proportion to n $\log_2$ n.

i.e. $\Theta$ (n $\log_2$ n).

**(c) Use quick sort algorithm to sort 15, 22, 30, 10, 15, 64, 1,3,9,52. Is it s stable sorting algorithm? Justify.**

**Ans.**

(15), 22, 30, 10, 15, 64, 1, 3, 9, (52)
9, 22, 30, 10, 15, 64, 1, 3, (15), 52
9, (15), 30, 10, 15, 64, 1, 3, 22, 52
9, 3, 30, 10, 15, 64, 1, (15), 22, 52
9, 3, (15), 10, 15, 64, 1, 30, 22, 52
9, 3, 1, 10, 15, 64, (15), 30, 22, 52
9, 3, 1, 10, 15, (15), 64 30, 22 52

first (1) (2) second

First : (1)

(9) 3, 1, 10 15
1, 3, (9), 10 15
1.1 1.2

Second: (2)

(64), 30, 22, 52
52, 30, 22, (64)
2.1

1.1: (1), 3
1.2: (10), 15

2.1: (52), 30, 22
22, 30, (52)

Result:

**1, 3, 9, 10, 15, 15, 22, 30, 52, 64**

**(d) Obtain the minimum number of entries that can be made in B-tree of order *m* and level *l*.**

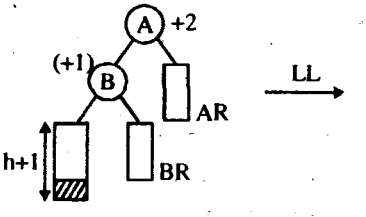
**Ans.** The order of B-tree is = m
The minimum no. of entries is = m/2
and maximum no. of entries is = m – 1

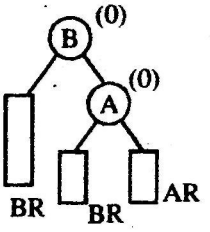
and having min. no. of child is = $\frac{m}{2} + 1$

and having max. no. of child is = m
At the level l;
If l is the leaf level, then every node at this level contain no child, and having almost m–1 keys or entries.
If l is the level of internal node, then every node at this level contain at most m child and having almost m-1 keys or entries.

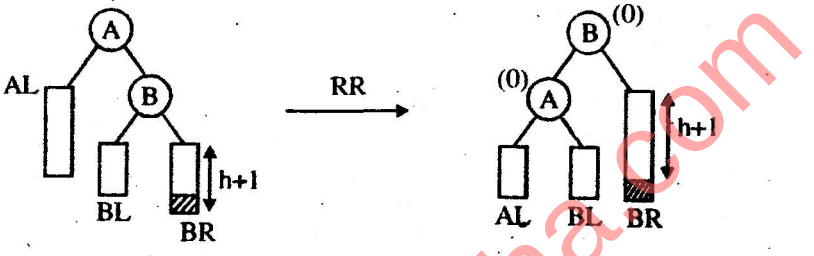
**(e) Discuss the different cases of rotations in AVL trees.**

**Ans.** (i) LL-Rotation:

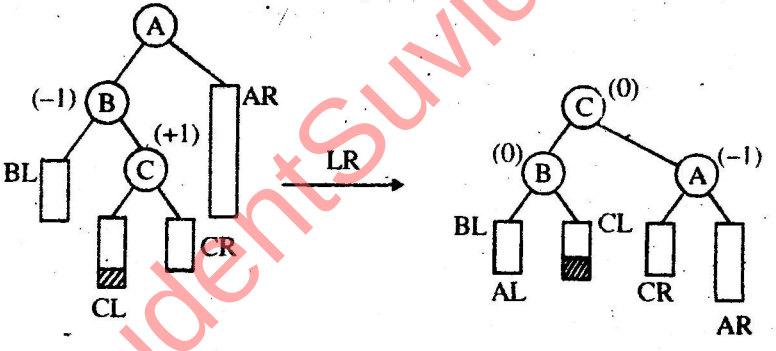




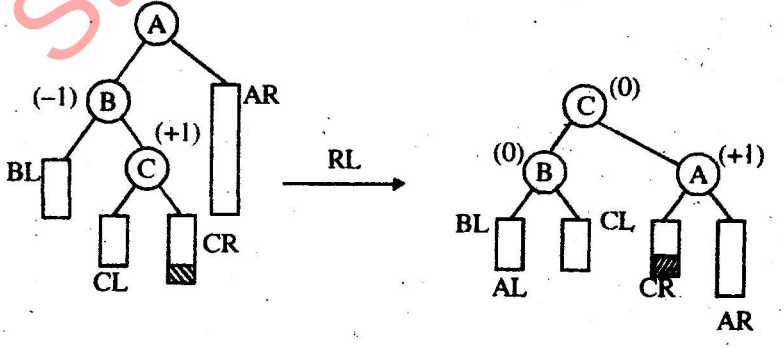

(ii) RR-Rotation




(iii) LR-Rotation




(iv) RL-Rotation



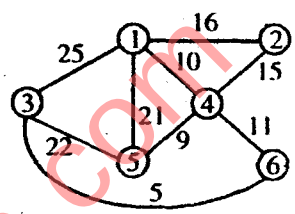
**(f) Write a function in C to insert an element into a binary search tree.**

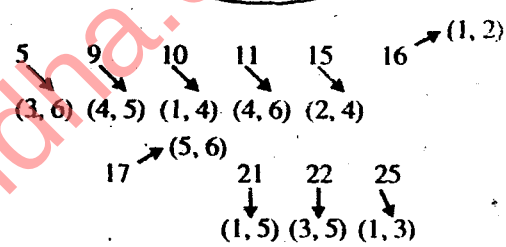
**Ans.** Insert (int item)

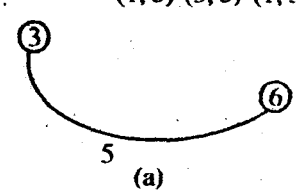
```
{
  struct node *tmp, *parent, *location;
  find (item, & parent, & location);
  if (location ! = NULL)
  {
    printf ("Item already present");
    return;
  }
  tmp = (struct node *) malloc (size of (struct
                  node);
  tmp → info = item;
  tmp → left = thread;
  tmp → right = thread;
  if (parent = = head)
  {
    head → left = link;
    head → left – ptr = tmp;
    tmp → left – ptr = head;
    tmp → right – ptr = head;
  }
  else
    if (item < parent → info)
      {
        tmp → left – ptr = parent → left – ptr;
        tmp → right – pir = parent;
        parent → left = link;
        parent → left – ptr = tmp;
      }
      else
      {
        tmp → left – = parent;
        tmp → right – ptr = parent → right – ptr;
        parent → right = link;
        parent→ right – ptr = tmp;
      }
  }
```

**5. Attempt any four parts of the following:** $5 \times 4 = 20$

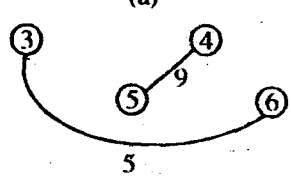
**(a) Use Kruskal's algorithm to determine the minimum spanning tree of the graph given below:**



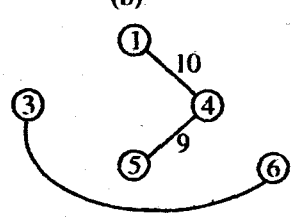



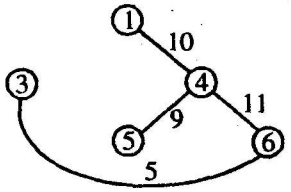

(a)



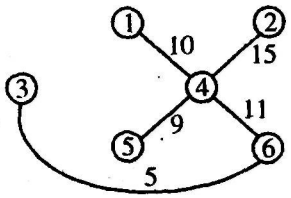

(b)



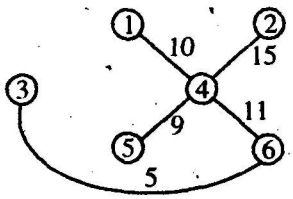

(c)




(d)




(e)



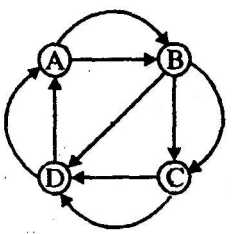

(f)

**(b) Explain the different techniques used to represent a graph in computer memory.**

**Ans. (i) Adjacency Matrix:** Adjacency matrix is the matrix, which keep, the information of adjacent nodes.

Arr [i] [j]

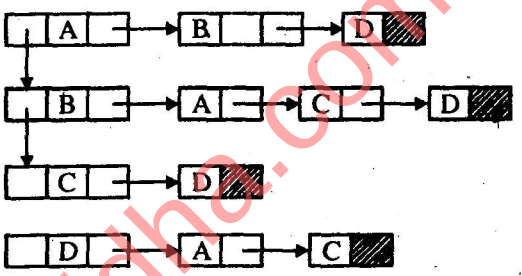
$$= \begin{cases} 1 \text{ if there is an edge from node i to j} \\ 0 \text{ otherwise} \end{cases}$$




(Graph)

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 0 | 1 |
| B | 1 | 0 | 1 | 1 |
| C | 0 | 0 | 0 | 1 |
| D | 1 | 0 | 1 | 0 |

(ii) **Adjacency List:** In adjacency list representation of graph, we will maintain two lists. First list will keep track of all nodes in the graph and second list will maintain a list of adjacency nodes for each node.




**(c) How are deletion of records handled in an indexed sequential file?**

**Ans.** Index is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations.

An index allows us to efficiently retrieve all records that satisfy search conditions on the search key field of the index.

We can also create additional indexes on a given collection of a data record.

- The data entry to refer to the records stored in an index file the data entry with search key value k, denoted as k* contains
- There are three main alternations
- A data entry k* is an actual data record
- A data entry is (k, rid) pair
- A data entry is (k, rid list > pair)

To delete a record that is specified using its rid we must identify the page that contains the record,

fetch it from disk, modify it and write it back. Depending on the file organization we may have to fetch modify and write back other pages as well.

**(d) Write a suitable algorithm to find the components of a graph.**

**Ans.** Components (G)

1. for each vertex v(–v [G]
2. do MAKE – set (v)
3. for each edge (u, v) ∈ ∈ (G)
4. do if FIND - SET (u) ≠ FIND-SET (v)
5. then UNION (u, v).

SAME - Component (u, v)

1. if FIND - SET (u) = FIND - SET (V)
2. then return TRUE
3. else return FALSE.

**(e) Discuss the factors involved in selecting a particular file organization.**

**Ans.** There are particular factors involving in selecting a particular file organization:

**Scan:** Fetch all records in the file. The pages of the file must be fetched from disk into the buffer pool. There is also a CPU overhead per record on the page.

**Accessing:** File organization must provide easy way of accessing from disk. There are no. of ways of accessing as sequential file accessing and direct accessing.

**Inserting:** To select a particular file structure anis factor must keep in mind that inserting into file must be easily done.

**Deletion:** Deletion must be also easily done in a efficient file organization and their should be efficient scheme for compaction or garbage collection of free space.

**Cost:** Cost should be minimum of a efficient file organization i.e., time of execution ex. CPU cost and network transition cost in data base (distributed).

**Modification:** There should be efficient mechanism for modification of file structure so that there is no damage in data structure in the disk.

**(f) Write short notes on any two:**

**(i) Direct file organization**

**(ii) Prims algorithm**

**(iii) Indexing.**

**Ans. Indexing:** An index is data structure that organize data records on disk to optimize certain kinds of retrieval operations:

An index allows us to efficiently retrieve all records that satisfy search condition on the search key field of index.

- The term data entry to refer to records stored in an index file.
- A data enter with search key value k denoted k*
- There are different file indeed file organization.
- **Clustered:** When file is organized so that ordering of data records in close to ordering of data entries in some index called clustered.
- **Primary index:** The index that uses one of the following rule.
  - (a) A data entry is a <k, rid) pair rid is record rid
  - (b) A data entry is a < k, rid - list> pair rid is record list
- Hash based indexing: In this technique records are organized using a technique called hashing In this approach files are grouped in buckets where bucket consist of a primary page. Bucket are retrieved through hash function.
- **Free- based :** In tree based indexing data are organized as a hierarchy way as tree structure.

(ii) Prims algorithm: Prims algorithm is also based on the generic MST algorithm. Prims alongwith has property that the edge in set A always form a single free. The free stage from an arbitrary root vertex r (grows untie the tree spans all the vertices

in v. At each step, a right edge is added to the tree A that connects A to an isolated vertex $g_A = (V,A)$. Here we add only edges which are safe for A, so A form a M.S.T.

(iii) Direct file organization: If the file records are numbered 0, 1, 2, ...r – 1 and the records in each block are numbered 0, 1... bfr–1, where bfr is the blocking factor, then the ith record of the file is records in block L [i/bfr] and is the (i mod bfr) the record in the block. Such a file is often called a relative or direct file. Because records can easily be accessed directly by their relative positions.

```
Node PTR maketre cmtnd
{
NODEPTRP;
P = getnodes;
P → info = n;
P → ief = NULLS
P → nght = NULL
P → rthread = TUUE:
rehm (p);
}
   Vard Setleft (NODEPTER P, int n)
   {
   NODEP TR Q;
   if (P = = NULL)
   em r (" vard inserha")
   else if (P → left ! = NULL)
    emr ("mvalid mserher);
   else
   {
   g = get node (i)
   g → info = x;
   p → left = q;
   q → left = NULL;
   q → right = p;
   q → rthred = TRUE;
   }
   }
Verd setnght CONDEPR P, int n)
{
NODEPTR Q, V;
if (P = = NULL)
cmr ("vard insrnim");
else if C:P → rlhread)
emr ("invalrd inserher)'
else
{
g = get node():
g → m go = n;
r = p → right
p → rlhread = SALSE;
q → left = NULL;
q → right = r;
q → rthread = TRUE;
}
}
```