B.Tech.

Third Semester Examination

Data Structures Using 'C' (CSE-201-F)

Note : Attempt any five questions. All the questions carry equal marks.

Q. 1. (a) Write the algorithms for the following :

(i) Interchange the elements of position P and next (P) in a Singly Linked List.

(ii) Locating an element on a sorted list using Array representation. What is the running time of each of these algorithm ?

Ans.

```
#include<stdio.h>
#include<malloc.h>
void main(void)
{
  int p;
  struct list entry
  {
    int number
    struct list entry*next;
  }
  start,*node;
start next = NULL; /*empty list*/
node = &start; /*point to the start of the list*/
for(i=1; i<=10, i++)
  {
  node → next = (struct list entry*) malloc (size of struct list entry));
  node = node → next;
  node → number = 50 + i;
  node → next = NULL;
}
//display the list
node = start next;
while (node)
  {
  print ("/n%d", node node → number);
  node = node → next;
  }
}
```

```
int update (int*);
void display (int*);
void main( )
{
int stack[size];
int data;
char choice;
int q=0;
int top = -1;
do
{
printf("\n push → iupdate → uquit → q:");
do
{
fflush (stain);
choice = getchar( );
choice = tolower(choice);
}
while (iq);
}
```

Q. 3. (a) Write down the non-recursive algorithm for pre order traversal of a binary tree.

Ans. Preorder (Node)

Step 1 : [Do through step 3]
if Node ≠ NULL
output info [Node]

Step 2 : call preorder (left_child [Node]).

Step 3 : call preorder (Right_Child [Tacode])

Step 4 : Exit

Q. 3. (b) Write an algorithm to count the number of leaves in a binary tree.

Ans. create_tree(info_Node)

where

Node → Structure type variable to points both left and right child.

Info → Information for which we have to create node.

Step 1 : [check whether the tree is empty]
If Node = NULL
Node = create a node
Left_child(Node) = NULL
Right_child(Node) = NULL

Step 2 : [Test for the left child]

if info[Node] ≥ info

left_child [Node] = Call create_tree (info, left_child [Node])

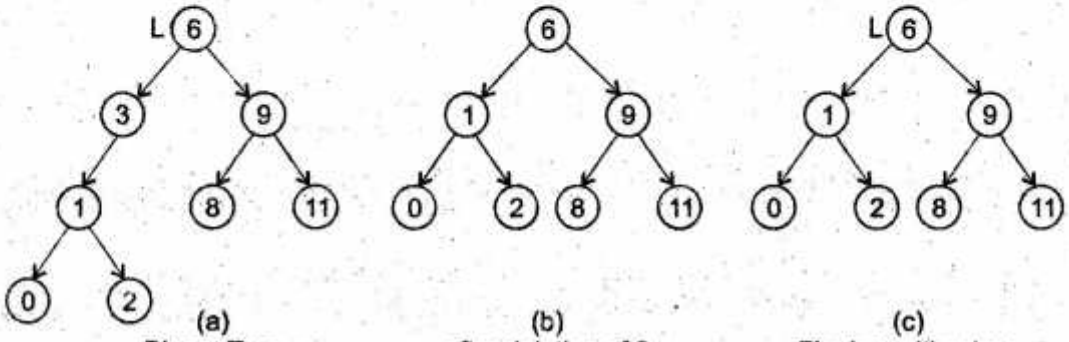
else

right_child[Node]= call create_tree (info, Right_child [Node])

Step 3 : return (Node)

Q. 4. (a) Write an algorithm to delete an element X from a binary search tree. Do the time analysis of your algorithm.

Ans.



Q. 4. (b) What are threaded Binary trees ? What are the applications ?

Ans. Because of the importance of linked binary trees, it is worth while to develop non-recursive algorithms, to manipulate them, & to study the time & space requirement of these algorithms. We shall find that by changing the NULL links in a binary tree to special links called threads, it is possible to perform traversals, insertions, & deletions without using either a stack or recursion.

In a threaded binary tree, each null right link is replaced by a special link to the succession of that node under in order traversal called a right thread.

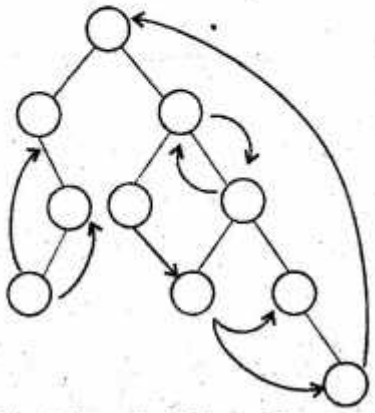


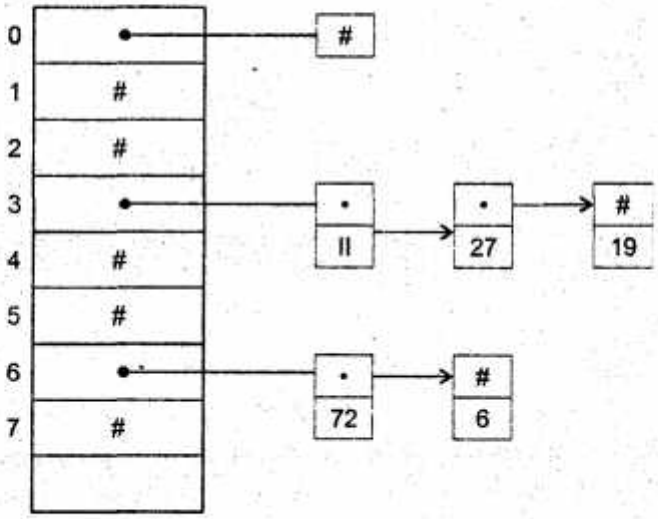
Fig. Threaded Binary Tree

Q. 5. (a) Explain how the balance is restored when an insertion into height balanced tree puts a node out of balance.

Ans. The problem associated with the sequential representation of binary tree can be overcome through the use of the linked list representation.

Q. 5. (b) Explain the various collision resolution techniques.

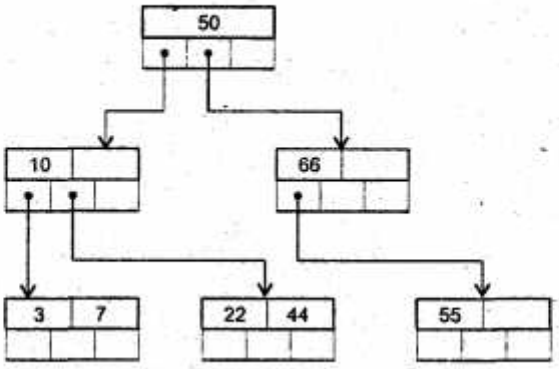
Ans. The problem of avoiding the collisions is the challenge in designing a good hash function. A good hash function minimizes collisions by spreading the elements uniformly throughout the array. But the minimization is very difficult job.



A hash tree

Q. 6. (a) Insert the following entries into an initially empty B-tree of order 5 :
a, g, f, b, k, c, h, n, j, d, r, i, s, x, e, l, m, t, u, v.

Ans.



3 way 3 tree

Q. 6. (b) What are the advantages and disadvantages of circular Linked List ?

Ans. A linked list in which last node points to the header node is called header/circular linked list. The chains do not indicate first/last nodes. In this case, external pointers provide a frame of reference because last node of a circular linked list does not contain the NULL pointer.

(i) Insertion

(ii) Deletion

(iii) Traversing

Q. 7. (a) Find out the time complexity of Quick sort. What happens if all the keys in the list are equal in case of Quick sort ?

Ans.

```
#include<stdio.h>
#include<stdlib.h>
void quick_sort (int array [ ], int first, int last)
{
int temp, low, high, list_separator;
    low = first;
    high = last;
    list_separator = array [(first + last)/2];
    do
    {
    while (array [low] < list_separator)
      low ++;
    while (array [high] > list_separator)
      high –;
    if low < = high
      {
      temp = array [low];
      array [low ++] = array [high];
      array [high –] = temp;
      }
```

Q. 7. (b) Write the algorithm for Insertion sort. How many key comparisons are made in its worst case ?

Ans. insertion_sort (l, n)

where, l → represents the list of elements

N → represents number of elements in the list.

Step 1 : [initialize]

l [0] = – 0

Step 2 : Repeat through step 3 through 5 for

i = 1, 2, 3, 4.....n.

Step 3 : (i) temp = l [i]

(ii) pointer = i -1

Step 4 : while [temp < = L (pointer)]

(i) l[pointer + 1] = l[pointer],

(ii) pointer = pointer – 1

Step 5 : l[pointer] = temp

Step 6 : exit.

Q. 8. (a) What are connected components of a Graph ? Write a method to find out all connected components of a graph ?

Ans. Step 1 : Define an array B that store Boolean values, its size should be greater or equal to the number of vertices in the graph G.

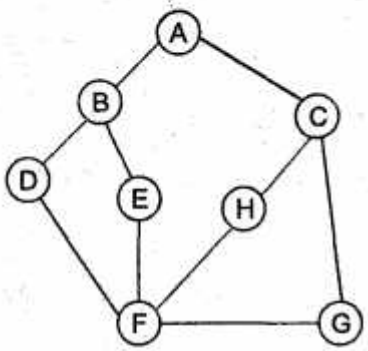
Step 2 : Initialize the array B to False.

Step 3 : if B[v] = false

process (v)

Step 4 : exit.

Q. 8. (b) A graph is shown below :



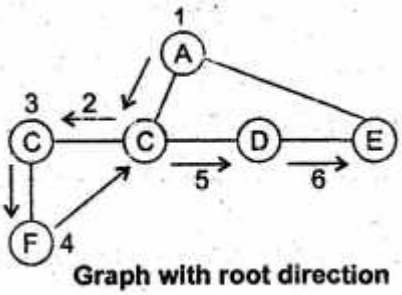
(i) Give Adjacency Matrix and List representation.

(ii) Give the Depth First Spanning tree and Breadth First Spanning tree.

Ans. Adjacency Matrix

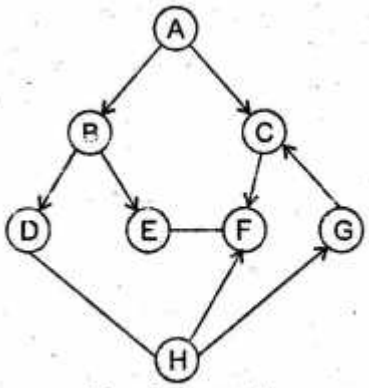
$$A = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{array} \begin{array}{c} \begin{array}{ccccc} V_1 & V_2 & V_3 & V_4 & V_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

DFS :



Graph with root direction

BFS :



Directed graph