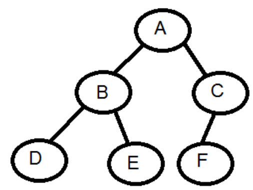
L. D. College Of Engineering
Assignment List
1^{ST} SEM ME -CE

M. E. SEMESTER: **I** **Computer Engineering**
Subject Name: **ADVANCE DATA STRUCTURES**
Subject Code: **3710215**

Note: To represent tree in the following programs, use set notations, e.g. The following tree in below figure can be represent as ((D)B(E))A((F)C). Again if node 'D' got right child 'F' then it is display as ((D(F))B(E))A((F)C).



Also output the inorder traversal of the resultant trees. The set representation is used to avoid the difficulty of printingpictorial representation.

Experiment 1 (Arrays, Linked List, Stacks, Queues, Binary Trees)

I. WAP to implement a 3-stacks of size 'm' in an array of size 'n' with all the basic operations such as IsEmpty(i), Push(i), Pop(i), IsFull(i) where 'i' denotes the stack number (1,2,3), m $\cong$ n/3. Stacks are not overlapping each other. Leftmost stack facing the left direction and other two stacks are facing in the right direction.

II. WAP to implement 2 overlapping queues in an array of size 'N'. There are facing in opposite direction to eachother. Give IsEmpty(i), Insert(i), Delete(i) and IsFull(i) routines for ith queue

III. WAP to implement Stack ADT using Linked list with the basic operations as Create(), Is Empty(), Push(), Pop(), IsFull() with appropriate prototype to a functions.

IV. WAP to implement Queue ADT using Linked list with the basic functions of Create(), IsEmpty(), Insert(), Delete() and IsFull() with suitable prototype to a functions

V. WAP to generate the binary tree from the given inorder and postorder traversal.

VI. WAP to generate the binary tree from the given inorder and preorder traversals.

Experiment 2 (Sorting & Searching Techniques)

I. WAP to implement Quick Sort on 1D array of Student structure (contains student_name, student_roll_no, total_marks), with key as student_roll_no. And count the number of swap performed.

II. WAP to implement Merge Sort on 1D array of Student structure (contains student_name, student_roll_no, total_marks), with key as student_roll_no. And count the number of swap performed.

III. WAP to implement Bubble Sort on 1D array of Employee structure (contains employee_name, emp_no, emp_salary), with key as emp_no. And count the number of swap performed.

IV. WAP to implement Binary search on 1D array of Employee structure (contains employee_name, emp_no, emp_salary), with key as emp_no. And count the number of comparison happened.

V. WAP to implement Bucket Sort on 1D array of Faculty structure (contains faculty_name, faculty_ID, subject_codes, class_names), with key as faculty_ID. And count the number of swap performed

VI. WAP to implement Radix Sort on 1D array of Faculty structure (contains faculty_name, faculty_ID, subject_codes, class_names), with key as faculty_ID. And count the number of swap performed.

Experiment 3 (Hashing)

I. WAP to store k keys into an array of size n at the location computed using a hash function, loc = key % n, where k<=n and k takes values from [1 to m], m>n. To handle the collisions use the following collision resolution techniques,

- Linear probing
- Quadratic probing

- Random probing
- Double hashing/rehashing
- Chaining

II. Implement the above program I using hash function from Division methods.

III. Implement the above program I using hash function from Truncation methods.

IV. Implement the above program I using hash function from Folding methods.

V. Implement the above program I using hash function from Digit analysis methods.

Experiment 4 (BST and Threaded Trees)

I. WAP for Binary Search Tree to implement following operations:

a. Insertion

b. Deletion

    i. Delete node with only child

    ii. Delete node with both children

c. Finding an element

d. Finding Min element

e. Finding Max element

f. Left child of the given node

g. Right child of the given node

h. Finding the number of nodes, leaves nodes, full nodes, ancestors, descendants.

II. WAP to implement Inorder Threaded Binary Tree with insertion and deletion operation.

III. WAP to implement Preorder Threaded Binary Tree with insertion and deletion operation.

IV. WAP to implement Postorder Threaded Binary Tree with insertion and deletion operation.

V. WAP to traverse given Inorder Threaded Binary Tree in inorder, preorder and postorder fashion.

VII WAP to traverse given Postorder Threaded Binary Tree in inorder, preorder and postorder fashion.

VIII. WAP to transform BST into Threaded Binary Tree.

Experiment 5 (AVL Trees and Red-Black Trees)

I. WAP for AVL Tree to implement following operations: (For nodes as integers)

a. Insertion: Test program for all cases (LL, RR, RL, LR rotation)

b. Deletion: Test Program for all cases (R0, R1, R-1, L0, L1, L-1)

c. Display: using set notation.

II. Implement the above program I for nodes as Student structure, with key as Student_ roll_no.

III. WAP to implement Red-Black trees with insertion and deletion operation for the given input data as Strings

IV. Implement the above program III for nodes as Employee structure, with key as emp_no.

V. WAP using function which computes the balance factor of any given node in a BST.

VI. WAP to transform BST into AVL trees and also count the number rotations performed.

VII. WAP to find whether the given BST is AVL tree or not.

VIII. WAP to convert BST into Red-Black trees.

IX. WAP to find the black height of any given node in Red-Black tree and find the black height of the Red-Balck tree.

Experiment 6 (B-Trees)

I. WAP to implement insertion, deletion, display and search operation in m-way B tree (i.e. a non-leaf node can have atmost m children) for the given data as integers (Test the program for m=3, 5, 7).

II. WAP to implement insertion, deletion, display and search operation in m-way B tree (i.e. a non-leaf node can have atmost m children) for the given data as strings (Test the program for m=3, 5, 7).

III. WAP to implement insertion, deletion, display and search operation in m-way B tree (i.e. a non-leaf node can have atmost m children) for the given data as Student structures(as given above), with key as student_ roll_no . (Test the program for m=3,5,7).

IV. WAP to implement insertion, deletion, display and search operation in m-way B tree (i.e. a non-leaf node can have atmost m children) for the given data as Employee structures (as given above), with key as emp_no. (Test the program for m=3, 5, 7).

V. WAP to implement insertion, deletion, display and search operation in m-way B tree (i.e. a non-leaf node can have atmost m children) for the given data as Faculty structures (as given above), with key as faculty_ID. (Test the program for m=3, 5, 7).

Experiment 7 (Min-Max Heaps, Binomial Heaps and Fibonacci Heaps)

I. WAP to implement insertion, deletion and display operation in Min-Max Heap for the given data as integers.

II. WAP to implement Make_Heap, Insertion, Find_Min, Extract_Min, Union, Decrease_Key and Delete_Key operations in Binomial Heap for the given data as strings.

III. WAP to implement Make_Heap, Insertion, Find_Min, Extract_Min, Union, Decrease_Key and Delete_Key operations in Fibonacci Heap for the given data as Student structures (contains student_name, student_roll_no, total_marks), with key as student_ roll_no.

IV. Implement the above program (I) of Min-Max heap for Employee structures (contains employee_name, emp_no, emp_salary), with key as emp_no.

V. Implement the above program (II) of Binomial Heap for Faculty structures (contains faculty_name, faculty_ID, subject_codes, class_names), with key as faculty_ID.

VI. Implement the above program (III) of Fibonacci Heap for strings.

Experiment 8 (Disjoint Sets)

I. WAP to implement Make_Set, Find_Set and Union functions for Disjoint Set Data Structure for a given undirected graph G(V,E) using the linked list representation with simple implementation of Union operation.

II. WAP to implement Make_Set, Find_Set and Union functions for Disjoint Set Data Structure for a given undirected graph G(V,E) using the linked list representation with weighted-union heuristic approach..

III. WAP to implement Make_Set, Find_Set and Union functions using Union by rank heuristic for Disjoint Set forest rooted trees representation.

IV. WAP to implement Make_Set, Find_Set and Union functions using Path compression heuristic for Disjoint Set forest rooted trees representation.

Experiment 9 (Graphs Algorithms)

I. WAP to perform topological sort on dag using depth first search.

II. WAP to generate minimum spanning tree in a connected, undirected weighted graph using Kuruskal's algorithm with disjoint set data structures.

III. WAP to generate minimum spanning tree in a connected, undirected weighted graph using Prims's algorithm with disjoint set data structures.

IV. WAP to find single-source shortest path in a weighted directed graph using Bellman-Ford algorithm

V. WAP to find single-source shortest path in a weighted dag using topological sort.

VI. WAP to implement Dijkstra's algorithm for single-source shortest path in a weighted directed graph using fibonacci heap.

VIII. WAP to find all-pairs shortest path using dynamic-programming algorithm based on matrix multiplication.

IX WAP to find all-pairs shortest path using Floyd-Warshall algorithm.

X. WAP to find all-pairs shortest path using Johnson's algorithm for sparse graphs.

XI WAP to print strongly connected components in a directed graph.

XII. WAP to find articulation points, bridges, and biconnected components usnig depth-first search in a connected, undirected graph.

Experiment 10 (String Matching)

I. WAP to perform string matching using naive algorithm

II. WAP to perform string matching using Rabin-Karp algorithm.

III. WAP to perform string matching using Finite Automata.

IV. WAP to perform string matching using Knuth-Morris-Pratt algorithm.

V. WAP to perform string matching using Boyer-Moore algorithm.