Society for Computer Technology & Research's

Pune Institute of Computer technology

Department of Information Technology

LAB MANUAL

Advanced Data Structures & Applications Laboratory

Class: - S.Y. B. Tech.

SCRT's Pune Institute of Computer Technology, Pune

Department of Information Technology

Lab Manual

Course Code & Name:

3403208: Advanced Data Structures and Applications Laboratory (ADSAL)

Class: SE Semester: IV

Prepared By:

Mr. Sachin D. Shelke
Ms. J. H. Jadhav
Mrs. D. P. Salapurkar
Ms. S. G. Gaikwad

Compiled By:
Mr. Sachin D. Shelke

VISION AND MISSION OF THE INSTITUTE

Vision:

Pune Institute of Computer Technology aspires to be the leader in higher technical education and research of international repute.

Mission:

To be leading and most sought after Institute of education and research in emerging engineering and technology disciplines that attracts, retains and sustains gifted individuals of significant potential.

VISION AND MISSION OF THE DEPARTMENT

Vision:

The department aspires to be a transformative force in Information Technology education and research, developing globally competitive professionals.

Mission:

1. Inculcate an environment of academic rigor in Information Technology education promoting critical thinking, creativity, and problem-solving skills.
2. Foster research and industry collaboration by promoting multiddisciplinary engagement among students and faculty, driving technological advancements to address societal challenges.
3. Cultivate skilled professionals effective communication, leadership, and ethical values.

PEO's OF THE DEPARTMENT

1. Have a strong background in science and mathematics and an ability to design and develop solutions for complex problems using modern tools in the field of Information Technology.
2. Attain professional competence by industry collaboration, research, and innovation to create solutions that drive technological progress and address societal and environmental concerns through lifelong learning.
3. Have ability to communicate effectively, work well in team, and to manage IT projects in multidisciplinary environment with ethical awareness.

PSO's OF THE DEPARTMENT

1. Develop computational solutions that integrate emerging fields such as Artificial Intelligence, Cloud Computing, Cybersecurity, and IoT to address the evolving needs of industry and society.
2. Ability to collaborate effectively in teams, leveraging strong interpersonal skills to deliver solutions for complex IT projects using cutting-edge methodologies, development tools, and techniques.

Annexure I: Knowledge and Attitude Profile (WK)

WK1: A systematic, theory-based understanding of the natural sciences applicable to the discipline and awareness of relevant social sciences.

WK2: Conceptually-based mathematics, numerical analysis, data analysis, statistics and formal aspects of computer and information science to support detailed analysis and modelling applicable to the discipline.

WK3: A systematic, theory-based formulation of engineering fundamentals required in the engineering discipline.

WK4: Engineering specialist knowledge that provides theoretical frameworks and bodies of knowledge for the accepted practice areas in the engineering discipline; much is at the forefront of the discipline.

WK5: Knowledge, including efficient resource use, environmental impacts, whole-life cost, reuse of resources, net zero carbon, and similar concepts, that supports engineering design and operations in a practice area.

WK6: Knowledge of engineering practice (technology) in the practice areas in the engineering discipline.

WK7: Knowledge of the role of engineering in society and identified issues in engineering practice in the discipline, such as the professional responsibility of an engineer to public safety and sustainable development.

WK8: Engagement with selected knowledge in the current research literature of the discipline, awareness of the power of critical thinking and creative approaches to evaluate emerging issues.

WK9: Ethics, inclusive behavior and conduct. Knowledge of professional ethics, responsibilities, and norms of engineering practice. Awareness of the need for diversity by reason of ethnicity, gender, age, physical ability etc. with mutual understanding and respect, and of inclusive attitudes.

PROGRAM OUTCOMES

PO1: Engineering Knowledge: Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

PO3: Design/Development of Solutions: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

PO4: Conduct Investigations of Complex Problems: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).

PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)

PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)

PO8: Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO9: Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences

PO10: Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one’s own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO11: Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8)

Course Objectives

The objective of this course is to provide students with

1. Ability to understand advanced data structures to solve complex problems in various domains.
2. Develop the logic to use appropriate data structure in logical and computational solutions.
3. Ability to develop applications using data structure algorithms.

Course Outcomes

At the end of the course, students will be able to

CO1: Implement and analyze Binary Search Tree (BST) and threaded binary tree operations and extend functionalities to compute identify structural properties.

CO2: Implement and analyze fundamental graph data structures and algorithms such as BFS, DFS, shortest path, minimum spanning tree, and graph traversal techniques.

CO3: Design and implement efficient string-matching algorithms such as Knuth-Morris-Pratt (KMP) to enhance search performance in large textual datasets and analyze its effectiveness.

CO-PO-PSO Mapping

CO NO.	*Course Outcomes*											*PSO*		
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**1**	**2**
CO1	Implement and analyze Binary Search Tree (BST) and threaded binary tree operations and extend functionalities to compute identify structural properties.	3	3	2	2	2	-	-	1	1	-	2	2	1
CO2	Implement and analyze fundamental graph data structures and algorithms such as BFS, DFS, shortest path, minimum spanning tree, and graph traversal techniques.	3	3	3	2	2	1	-	1	1	1	2	3	1
CO3	Design and implement efficient string-matching algorithms such as Knuth-Morris-Pratt (KMP) to enhance search performance in large textual datasets and analyze its effectiveness.	3	3	2	2	2	-	-	1	1	-	2	2	1
	Average	**3**	**3**	**2**	**2**	**2**	**1**	**-**	**1**	**1**	**1**	**2**	**2**	**1**

IMPORTANT GUIDELINES

- The laboratory assignments are to be submitted by students in the form of journals. The Journal consists of a table of contents (Index), and handwritten write-up of each assignment (Title, Aim, Problem Statement, Outcomes, Date of Implementation, Date of Submission, assessor's sign, Theory- Concept, algorithms, printouts of the code written using coding standards, sample test cases etc.)
- Practical Examination will be based on the assignment conducted in this course with multiple different applications.
- The candidate is expected to know the theory involved in the experiment.
- The practical examination should be conducted if journal of the candidate is completed inall respects and certified by the faculty concerned and head of the department.

Guidelines for Lab /TW Assessment

- Examiners will assess the term work based on performance of students considering the parameters such as timely conduction of practical assignment, methodology adopted for implementation of practical assignment, timely submission of assignment in the form of handwritten write-up along with results of implemented assignment, attendance etc.
- Examiners will judge the understanding of the practical performed in the examination by asking some questions related to theory & implementation of experiments he/she has carriedout.
- Appropriate knowledge of usage of software and hardware such as compiler, debugger, coding standards, algorithm to be implemented etc. should be checked by the concerned faculty member(s).

3403208: ADVANCED DATA STRUCTURES AND APPLICATIONS LABORATORY

Teaching Scheme:	Practical: 4 Hours/Week	**Credits: 02**
Examination Scheme:	CIE (TW): 25Marks ESE (PR): 25 Marks	

Prerequisites: Students should have prior knowledge of

- **Programming Languages:** JAVA
- **Concepts:** Basic Object-Oriented Programming (OOP)

Assig No.	Title and Description of Assignment	Hrs.	CO
	Practice Assignment • Print "Hello, World!" • Check whether a number is even or odd • Find the largest of three numbers • Check whether a number is positive, negative, or zero • Simple calculator (Menu-driven) • Check whether a year is a leap year • Print the multiplication table of a number • Find the sum of first N natural numbers • Reverse a number • Check whether a number is prime • Find the factorial of a number • Generate Fibonacci series • Find the sum of digits of a number • Print patterns using loops (e.g., *, triangle, number Patterns) • Find the largest element in an array • Find the smallest element in an array • Search an element in an array (linear search) • Sort an array of integers (basic bubble/selection sort) • Check whether a string is a palindrome • Count vowels and consonants in a string • Convert string to uppercase and lowercase • Create a simple menu-driven program (loops + switch) • Create a 'Student' class and store/display details (intro to OOP)	08	

1	**Binary Search Tree:** Implement binary search tree and perform following operations: a) Insert (Handle insertion of duplicate entry) b) Delete c) Search d) Display tree (Traversal) e) Display - Depth of tree f) Display - Mirror image g) Create a copy h) Display all parent nodes with their child nodes i) Display leaf nodes j) Display tree level wise (Note: Insertion, Deletion, Search and Traversal are compulsory, from rest of operations, perform Any three)	08	CO1
2	**Heap:** A software company is developing a new job scheduler that efficiently manages different processes. The goal is to minimize average waiting time by giving priority to shortest time tasks First use a Min-Heap (Priority Queue) for implementation. (Or giving priority to the largest time task First use a Max Heap Priority Queue for implementation.)	06	CO2
3	**Threaded Binary Tree:** Consider a binary tree and implement In-order Threaded Binary Tree and traverse it in In-order and Pre-order.	06	CO1
4	**Graph:** A city transportation department is developing a navigation system to help users find efficient routes between different locations. Represent the city map as a graph, having node as each location (place) and Each road connecting two locations is an edge. Explore possible routes using Depth-First Search (DFS) and Breadth-First Search (BFS).	06	CO3
5	**Graph- Minimum Spanning Tree:** Represent a graph of your college campus using adjacency list /adjacency matrix. Nodes should represent the various departments/institutes and links should represent the distance between them. Find the minimum spanning tree using a) Kruskal's algorithm. B) Prim's algorithm. Analyze the above two algorithms for space and time complexity.	06	CO3
6	**Graph- Shortest Path Algorithm:** Represent a graph of the city using an adjacency matrix /adjacency list. Nodes should represent the various landmarks and links should represent the distance between them. Find the shortest path using Dijkstra's algorithm from single source to all destinations. Analyze the implemented algorithm for space and time complexity.	06	CO3
7	**KMP for string matching:** A library management system is developing a text search engine to help users quickly locate books, research papers, and articles by searching for specific keywords within large texts. Traditional brute-force string matching algorithms are inefficient for searching in large datasets, leading to slower search results. Implement KMP for string matching in a text search engine.	06	CO6

This is for Student's Journal

Assignment write-up should be in this format

Assignment No:

Title:

Aim:

Objective:

Problem Statement:

Theory / Procedure / Diagrams / Circuits:

Algorithm / Methods / Steps: (if applicable)

Conclusion:

Assignment No: 1

Title: Binary Search Tree

Aim: To implement a Binary Search Tree (BST) and perform the fundamental operations such as insertion (with handling of duplicate entries), deletion, searching, and tree traversals.

Objective:

1. To understand the concept and structure of a Binary Search Tree (BST) and its use as a hierarchical data structure.
2. To implement basic BST operations such as insertion (with duplicate handling), deletion, searching and traversal.
3. To apply recursive and iterative techniques for performing different tree operations and traversals.

CO Mapped: CO1

Problem Statement:

Implement binary search tree and perform following operations: a) Insert (Handle insertion of duplicate entry) b) Delete c) Search d) Display tree (Traversal) e) Display - Depth of tree f) Display - Mirror image g) Create a copy h) Display all parent nodes with their child nodes i) Display leaf nodes j) Display tree level wise
Note: Insertion, Deletion, Search and Traversal are compulsory, from rest of operations, perform Any three

Expected Outcome:

1. Students will be able to implement a Binary Search Tree (BST) and perform operations such as insertion, deletion, searching and traversal.
2. Students will understand and apply recursive techniques to perform various BST operations.
3. Students will be able to analyse the behaviour of BST operations with respect to the structure and height of the tree.

Pre-requisites:

1. Understanding of arrays, structures and dynamic memory allocation.
2. Familiarity with recursion and iterative program logic.
3. Basic understanding of data structures concepts such as stacks, queues and linked lists.

Suggested Study Material: (Blogs / Videos / Courses / Web Sites / Books / e-Books)

- https://www.geeksforgeeks.org/dsa/binary-search-tree-data-structure/?utm_source=chatgpt.com
- https://www.programiz.com/dsa/binary-search-tree?utm_source=chatgpt.com

- https://www.digitalocean.com/community/tutorials/binary-search-tree-bst-search-insert-remove?

Implementation Requirements: (Platform / Software)

Platform: IntelliJ IDEA

Input: Nodes- 6 to 8, Tree height-3 [This is the minimum expected input]

Theory / Procedure / Diagrams:

- Concepts of Binary Tree, Binary Search Tree (BST)
- BST Properties
- BST Operations with example [Insert, Search, Delete, Traversal etc.]
- BST Applications

Algorithm / Methods / Steps:

➤ **Insert Node**

```
INSERT (ROOT, ITEM):  // Recursive Algorithm

  IF ROOT == NULL:
    ROOT = new node (ITEM)
    RETURN ROOT

  IF ITEM == ROOT->data:
    PRINT "Duplicate key - insertion not allowed"
    RETURN ROOT

  IF ITEM < ROOT->data:
    ROOT->left = INSERT(ROOT->left, ITEM)
  ELSE
    ROOT->right = INSERT(ROOT->right, ITEM)

  RETURN ROOT


INSERT_BST (ROOT, ITEM) // Non-Recursive Algorithm

Create a new node NEW
Set NEW->data = ITEM
Set NEW->left = NULL and NEW->right = NULL

If ROOT = NULL then
    ROOT = NEW
    Exit

Set PTR = ROOT
Set PREV = NULL
```

```
While PTR is not NULL do
    PREV = PTR
    If ITEM = PTR->data then
        Print "Duplicate key — insertion not allowed"
        Free NEW
        Exit
    Else If ITEM < PTR->data then
        PTR = PTR->left
    Else
        PTR = PTR->right
End While

If ITEM < PREV->data then
    PREV->left = NEW
Else
    PREV->right = NEW

End Algorithm
```

- **Delete Node**

```
Algorithm DELETE_BST (ROOT, KEY)

If ROOT = NULL then
    Print "Tree is empty"
    Exit

Set PTR = ROOT
Set PARENT = NULL

// Step 1: Search for the node to delete
While PTR is not NULL AND PTR->data ≠ KEY do
    PARENT = PTR
    If KEY < PTR->data then
        PTR = PTR->left
    Else
        PTR = PTR->right
End While

If PTR = NULL then
    Print "Key not found"
    Exit

// Step 2: Case 1 — Node with no child (Leaf Node)
If PTR->left = NULL AND PTR->right = NULL then
    If PARENT = NULL then
        ROOT = NULL
    Else If PARENT->left = PTR then
        PARENT->left = NULL
    Else
```

```
            PARENT->right = NULL
        Free PTR
        Exit

    // Step 3: Case 2 — Node with one child
    If PTR->left = NULL OR PTR->right = NULL then
        If PTR->left ≠ NULL then
            CHILD = PTR->left
        Else
            CHILD = PTR->right

        If PARENT = NULL then
            ROOT = CHILD
        Else If PARENT->left = PTR then
            PARENT->left = CHILD
        Else
            PARENT->right = CHILD

        Free PTR
        Exit

    // Step 4: Case 3 — Node with two children
    Set SUCCESSOR_PARENT = PTR
    Set SUCCESSOR = PTR->right

    While SUCCESSOR->left ≠ NULL do
        SUCCESSOR_PARENT = SUCCESSOR
        SUCCESSOR = SUCCESSOR->left
    End While

    PTR->data = SUCCESSOR->data

    If SUCCESSOR_PARENT->left = SUCCESSOR then
        SUCCESSOR_PARENT->left = SUCCESSOR->right
    Else
        SUCCESSOR_PARENT->right = SUCCESSOR->right

    Free SUCCESSOR
    End Algorithm
```

- **Search Node**

```
    Algorithm SEARCH_BST (ROOT, KEY)

    If ROOT = NULL then
        Print "Tree is empty"
        Exit

    Set PTR = ROOT
```

```
While PTR is not NULL do
    If KEY = PTR->data then
        Print "Key found"
        Exit

    Else If KEY < PTR->data then
        PTR = PTR->left
    Else
        PTR = PTR->right
End While

Print "Key not found"
End Algorithm
```

- **Display Tree**

```
Algorithm INORDER(ROOT) //In order Display

If ROOT is not NULL then
    INORDER(ROOT->left)
    Print ROOT->data
    INORDER(ROOT->right)
End Algorithm

Algorithm PREORDER(ROOT) //Pre order Display

If ROOT is not NULL then
    Print ROOT->data
    PREORDER(ROOT->left)
    PREORDER(ROOT->right)
End Algorithm

Algorithm POSTORDER(ROOT) //Post order Display

If ROOT is not NULL then
    POSTORDER(ROOT->left)
    POSTORDER(ROOT->right)
    Print ROOT->data
End Algorithm
```

- **Depth of the Tree**

```
Algorithm Height(root)

If root = NULL
   return 0

leftHeight = Height (root → left)
rightHeight = Height (root → right)
```

```
return 1 + maximum (leftHeight, rightHeight)
End Algorithm
```

- **Mirror Image**

```
Algorithm Mirror(root)

If root = NULL
    return

Swap (root → left, root → right)

Mirror (root → left)
Mirror (root → right)
End Algorithm
```

- **Create Copy**

```
Algorithm CopyTree(root)

If root = NULL
    return NULL

Create a new node temp
temp → data = root → data

temp → left = CopyTree (root → left)
temp → right = CopyTree (root → right)
return temp
End Algorithm
```

- **Display Parent Nodes**

```
Algorithm DisplayParentWithChildren (root)

If root = NULL
    return

If root → left ≠ NULL OR root → right ≠ NULL
    Print "Parent =", root → data
    If root → left ≠ NULL
        Print " Left Child =", root → left → data
    Else
        Print " Left Child = NULL"
    If root → right ≠ NULL
        Print " Right Child =", root → right → data
    Else
        Print " Right Child = NULL"

DisplayParentWithChildren (root → left)
DisplayParentWithChildren (root → right)
End Algorithm
```

➢ **Display Leaf Nodes**

```
Algorithm DisplayLeafNodes (root)

If root = NULL
   return

If root → left = NULL AND root → right = NULL
   Print root → data
   return

DisplayLeafNodes (root → left)
DisplayLeafNodes (root → right)
End Algorithm
```

➢ **Level wise Display**

```
Algorithm LevelOrderTraversal(root)

If root = NULL
   return

Create an empty queue Q
Enqueue root into Q

While Q is not empty
   node = Dequeue(Q)
   Print node → data

   If node → left ≠ NULL
      Enqueue(node → left)

   If node → right ≠ NULL
      Enqueue(node → right)

End While
End Algorithm
```

Test Cases:

Test algorithms on:

1. Tree of at least 6-8 nodes
2. Duplicate elements for Insertion operation
3. Successful and unsuccessful cases for insertion, deletion, and search
4. Null tree

Expected Output: (Results / Visualization)

1. Formatted display (Tree like structure)
2. Appropriate feedback/ messages

Inference:

- BST allows efficient insertion, deletion, and search operations.
- Traversals provide different ways to access tree data; in order gives sorted output.
- Tree operations like depth, mirror, copy, and leaf/parent display help understand tree structure and recursion.

Oral questions:

1. What is a Binary Search Tree (BST) and how does it differ from a regular binary tree?
2. What is the property of a BST?
3. Can a BST contain duplicate values? How can you handle duplicates?
4. Explain the algorithm for inserting a node into a BST.
5. Explain the algorithm for deleting a node from a BST.
6. How do you search for an element in a BST? What is its time complexity?
7. What are the different types of tree traversals? Explain each briefly.
8. What is the difference between in-order, preorder, and post-order traversal?
9. How can you display all leaf nodes of a BST?
10. How do you calculate the depth (height) of a BST?
11. What is a mirror image of a BST and how can it be created?
12. How can you create a copy of a BST?
13. Explain how to display all parent nodes with their child nodes in a BST.
14. How do you perform level-wise (breadth-first) traversal of a BST?
15. What are the advantages and applications of using a BST over other data structures?

Assignment No: 2

Title: Heap (Priority Queue)

Aim: To design and implement a job scheduler using Heap (Priority Queue) that efficiently manages processes by prioritizing tasks based on execution time to minimize average waiting time.

Objectives:

1. To understand the concept of Heap data structure and its applications in priority-based scheduling.
2. To implement Min-Heap and Max-Heap using Priority Queue.
3. To analyze how task scheduling efficiency improves using Heap-based priority mechanisms.

CO Mapped: CO2

Problem Statement:

A software company is developing a new job scheduler that efficiently manages different processes. The goal is to minimize average waiting time by giving priority to the shortest time task first using a Min-Heap (Priority Queue). Alternatively, priority can be given to the largest time task first using a Max-Heap (Priority Queue).

Implement the following operations using a Heap:

- Insert a process with execution time
- Delete the highest priority process
- Display processes based on priority
- Peek the highest priority process

Expected Outcome:

- Students will be able to implement Min-Heap and Max-Heap using Priority Queue.
- Students will understand priority-based process scheduling.
- Students will be able to analyze heap operations and their time complexity.

Pre-requisites:

- Basic understanding of arrays
- Knowledge of trees and complete binary trees
- Familiarity with priority queues and scheduling concepts

Suggested Study Material:

- https://www.geeksforgeeks.org/heap-data-structure/
- https://www.programiz.com/dsa/heap-data-structure
- https://www.tutorialspoint.com/data_structures_algorithms/heap_data_structure.htm

Implementation Requirements:
Platform: IntelliJ IDEA
Input: 6–8 processes with different execution times

Theory / Procedure / Diagrams:

- Heap Data Structure
- Min-Heap and Max-Heap properties
- Priority Queue concept
- Heap applications in job scheduling

Algorithm / Methods / Steps:

1. Min-Heap Insertion Algorithm
```
INSERT(heap, element):
    Add element at the end of heap
    While element < parent:
        Swap element with parent
```

2. Delete from Min-Heap
```
DELETE_MIN(heap):
    Remove root element
    Replace root with last element
    Heapify down to maintain heap property
```

3. Max-Heap Insertion Algorithm
```
INSERT(heap, element):
    Add element at the end of heap
    While element > parent:
    Swap element with parent
```

4. Delete from Max-Heap
```
DELETE_MAX(heap):
  Remove root element
  Replace root with last element
  Heapify down
```

Test Cases:

- Insert processes with varying execution times
- Delete highest priority process

- Check scheduler output for Min-Heap and Max-Heap
- Empty heap condition

Expected Output:

- Process execution order based on priority
- Correct scheduling with minimum waiting time
- Appropriate messages for empty heap or invalid operations

Inference:

Heap-based priority queues offer an efficient method for managing process scheduling. Min-Heap minimises waiting time by executing the shortest tasks first, while Max-Heap prioritises longer tasks when required. Heap operations ensure optimal performance with a time complexity of O (log n).

Oral Questions:

1. What is a Heap data structure?
2. Difference between Min-Heap and Max-Heap?
3. Why is Heap used in job scheduling?
4. What is a Priority Queue?
5. What is the time complexity of heap insertion and deletion?
6. How does Min-Heap reduce average waiting time?

Assignment No: 3

Title: Threaded Binary Tree

Aim: To implement an In-order Threaded Binary Tree and perform In-order and Pre-order traversals without using recursion or stack.

Objectives:

- To understand the concept of Threaded Binary Tree (TBT).
- To implement an In-order Threaded Binary Tree.
- To perform In-order and Pre-order traversal using threads.
- To analyze the advantages of threaded binary trees over normal binary trees.

CO Mapped: CO1

Problem Statement:

Consider a binary tree and implement an **In-order Threaded Binary Tree**. Perform the following operations:

- Create an In-order Threaded Binary Tree
- Traverse the tree using In-order traversal
- Traverse the tree using Pre-order traversal

Expected Outcome:

- Students will be able to implement In-order Threaded Binary Tree.
- Students will understand traversal of binary trees without recursion or stack.
- Students will be able to analyze memory-efficient tree traversal techniques.

Pre-requisites:

- Knowledge of binary trees
- Understanding of pointers and dynamic memory allocation
- Familiarity with tree traversal techniques

Suggested Study Material:

- https://www.geeksforgeeks.org/threaded-binary-tree/
- https://www.scaler.com/topics/threaded-binary-tree
- http://www.btechsmartclass.com/data_structures/threaded-binary-trees.html

Implementation Requirements:
Platform: IntelliJ IDEA
Input: Binary tree with 6–8 nodes

Theory / Procedure / Diagrams:

- Binary Tree concepts
- Threaded Binary Tree (TBT)
- In-order Threading
- Advantages of Threaded Binary Tree
- In-order and Pre-order traversal using threads

Algorithm / Methods / Steps:

Create In-order Threaded Binary Tree

```
Create node with data
If left child is NULL:
    left pointer points to inorder predecessor
    mark leftThread = true
If right child is NULL:
    right pointer points to inorder successor
    mark rightThread = true
```

In-order Traversal of Threaded Binary Tree

```
Set current = leftmost node
While current is not NULL:
    Print current data
    If rightThread is true:
        current = current.right
    Else:
        current = leftmost(current.right)
```

Pre-order Traversal of Threaded Binary Tree

```
Set current = root
While current is not NULL:
    Print current data
    If leftThread is false:
        current = current.left
    Else:
        current = current.right
```

Test Cases:

- Create a threaded binary tree with a minimum of 6 nodes
- Perform In-order traversal
- Perform Pre-order traversal

- Empty tree case

Expected Output:

- Correct In-order traversal sequence
- Correct Pre-order traversal sequence
- Traversal without recursion or stack

Inference:

Threaded Binary Trees improve traversal efficiency by replacing NULL pointers with threads. This eliminates the need for recursion or an auxiliary stack, making tree traversal faster and more memory-efficient.

Oral Questions:

1. What is a Threaded Binary Tree?
2. What is In-order Threading?
3. Difference between normal binary tree and threaded binary tree?
4. Why are threads used in binary trees?
5. How is In-order traversal done without recursion?
6. What are the advantages of Threaded Binary Tree?

Assignment No: 4

Title: Graph

Aim: To implement a city navigation system by representing the city map as a graph and exploring possible routes between locations using **Depth-First Search (DFS)** and **Breadth-First Search (BFS)** algorithms.

Objectives:

1. To understand the concept **Depth-First Search (DFS) and Breadth-First Search (BFS).**
2. To implement basic BST operations such as insertion (with duplicate handling), deletion, searching and traversal.
3. To apply recursive and iterative techniques for performing different tree operations and traversals.

CO Mapped: CO3

Problem Statement:

A **city transportation department** is developing a **navigation system** to help users find efficient routes between different locations. Represent the city map as a **graph**, having node as each **location (place) and** Each **road connecting two locations** is an **edge**. **Explore possible routes** using **Depth-First Search (DFS) and Breadth-First Search (BFS).**

Expected Outcome:

1. Students will be able to implement and to traverse locations using DFS and BFS.
2. Students will Identify of all possible routes using DFS and the shortest route using BFS.
3. Students will be able to represent a city map as a graph.

Pre-requisites:

1. Understanding of Graph theory concepts (graph, node, edge).
2. Adjacency list or adjacency matrix representation.
3. Basic understanding of data structures concepts such as stacks, queues.

Suggested Study Material: (Blogs / Videos / Courses / Web Sites / Books / e-Books)

- https://www.geeksforgeeks.org/dsa/graph-data-structure/
- https://www.programiz.com/dsa/graph-bfs
- https://www.digitalocean.com/community/tutorials/breadth-first-search-depth-first-search-bfs-dfs

Implementation Requirements: (Platform / Software)

Platform: IntelliJ IDEA
Input: Nodes- 6 to 8

Theory / Procedure / Diagrams:

- Introduction to Graph
- Graph terminology
- Different representations of Graph

Algorithm / Methods / Steps:

➢ **Depth-First Search (DFS)**

```
DFS(Graph, start):
    create an empty set visited
    call DFS_Visit(start)

DFS_Visit(node):
    mark node as visited
    display node
    for each adjacent node neighbor:
        if neighbor is not visited:
            DFS_Visit(neighbor)
```

➢ **Breadth-First Search (BFS)**

```
BFS(Graph, start):
    create an empty set visited
    create an empty queue Q
    mark start as visited
    enqueue start into Q

    while Q is not empty:
        node = dequeue Q
        display node
        for each adjacent node neighbor:
            if neighbor is not visited:
                mark neighbor as visited
                enqueue neighbor into Q
```

Test Cases:

Test algorithms on:

1. Tree of at least 6-8 nodes
2. Duplicate elements for Insertion operation
3. Successful and unsuccessful cases for insertion, deletion, and search
4. Null tree

Expected Output: (Results / Visualization)

1. Formatted display (Tree like structure)
2. Appropriate feedback/ messages

Inference:

1. BFS is more suitable for finding efficient routes, while DFS helps in exploring all possible paths.
2. DFS explores deep routes first and is useful for checking route availability.
3. BFS Explores nearest locations first and is best for navigation systems.

Oral questions:

1. What is a graph?
2. Why is a city map represented as a graph?
3. What is the difference between DFS and BFS?
4. Which data structure is used in DFS?
5. Which data structure is used in BFS?
6. Which algorithm finds the shortest path?
7. Is DFS suitable for navigation systems? Why or why not?
8. Can BFS be used in weighted graphs?
9. What is an adjacency list?
10. Give a real-life application of BFS.

Assignment No: 5

Title: Graph- Minimum Spanning Tree

Aim: To implement Minimum Spanning tree using Prims and Kruskal's Algorithm

Objectives:

1. To study Graph theory.
2. To study different graph traversal methods.
3. To understand the real time applications of graph theory.
4. To Implement MST using Prims and Kruskal's algorithm

CO Mapped: CO3

Problem Statement:

Represent a graph of your college campus using adjacency list/adjacency matrix. Nodes should represent the various departments/institutes and links should represent the distance between them. Find the minimum spanning tree using a) Kruskal's algorithm. B) Prim's algorithm.
Analyze the above two algorithms for space and time complexity.

Expected Outcome:

1. Students will be able to implement Prim's and Kruskal's algorithms, a **Minimum Spanning Tree (MST)** of a given weighted, connected graph is obtained.

2. Students will be able to analyse that both algorithms produce an optimal solution, though their execution steps and efficiency vary based on the graph's structure.

Pre-requisites:

- Knowledge of graphs (vertices, edges, weights)
- Understanding of greedy algorithms
- Familiarity with adjacency list and adjacency matrix
- Basic knowledge of sorting and priority queues

Suggested Study Material: (Blogs / Videos / Courses / Web Sites / Books / e-Books)

- https://www.geeksforgeeks.org/dsa/prims-minimum-spanning-tree-mst-greedy-algo-5/
- https://www.geeksforgeeks.org/dsa/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/
- https://www.programiz.com/dsa/prim-algorithm
- https://www.programiz.com/dsa/kruskal-algorithm

Implementation Requirements: (Platform / Software)
Platform: IntelliJ IDEA
Input: Nodes- 6 to 8

Theory / Procedure / Diagrams:

- Introduction to Graph
- Graph terminology
- Different representations of Graph
- Spanning Tree
- Minimal Spanning Tree
- Prim's Algorithm
- Kruskal's Algorithm
- Applications of Spanning Tree

Algorithm / Methods / Steps:

➢ **Prim's Algorithm**

- Start with a graph having V vertices.
- Select any vertex as the starting vertex.
- Initialize:
  key[] → store minimum edge weight (initialize to ∞).
  mstSet[] → track vertices included in MST (initialize to false).
- Set the key value of the starting vertex to 0.
- Repeat until all vertices are included:
- Select the vertex with **minimum key value** not yet in MST.
- Include this vertex in MST.
- Update key values of its adjacent vertices.
- The edges selected form the **Minimum Spanning Tree**.

```
PRIMS(weight, V)

KNOWN[1..V], COST[1..V], PREV[1..V]
Step 1: For i = 1 to V
        KNOWN[i] = 0
        COST[i]  = ∞
        PREV[i]  = -1
Step 2: COST[1] = 0          // Start from vertex 1
        TOTAL_COST = 0
Step 3: Repeat V times
        a) Find vertex u such that
           KNOWN[u] = 0 and COST[u] is minimum
        b) KNOWN[u] = 1
           TOTAL_COST = TOTAL_COST + COST[u]
```

```
    c) For v = 1 to V
       If weight[u][v] > 0 AND KNOWN[v] = 0 AND weight[u][v] < COST[v]
          COST[v] = weight[u][v]
          PREV[v] = u
Step 4: Print MST edges using PREV[]
    For i = 2 to V
       Print (PREV[i], i)
Step 5: Print TOTAL_COST
```

Trace of Prim 's algorithm for graph G1 starting from vertex 1

Step No.	Set A	Set (V-A)	Min cost Edge (u, v)	Cost	Set B
Initial	{1}	{2,3,4,5,6,7}	--	--	{}
1	{1,2}	{3,4,5,6,7}	(1, 2)	1	{(1, 2)}
2	{1, 2, 3}	{4, 5, 6, 7}	(2, 3)	2	{(1,2},(2,3)}
3	{1,2,3,5}	{4, 6, 7}	(2, 5)	4	{(1,2),(2,3),(2,5)}
4	{1,2,3,5,4}	{6, 7}	(1,4)	4	{(1,2),(2,3),(2,5),(1,4)}
5	{1,2,3,5,4,7}	{6}	(4,7)	4	{(1,2),(2,3),(2,5),(1,4),(4,7)}
6	{1,2,3,5,4,7.6 }	{ }	(7,6)	3	{(1,2),(2,3),(2,5),(1,4),(4,7),(7,6)}
		Total Cost		17	

Thus, the minimum spanning tree for graph G1 is : A = { 1,2,3,4,5,7,6}
B={(1,2),(2,3),(2,5),(1,4),(4,7),(7,6)}, total Weight: 1+2+4+3+4+3=17

- **Kruskal's Algorithm**

1. Sort all edges in **increasing order of weight**.
2. Initialize each vertex as a separate set.
3. Pick the smallest edge.
4. If the selected edge forms a **cycle**, discard it.
5. Otherwise, include it in MST and union the sets.
6. Repeat until MST contains V - 1 edges.

```
KRUSKAL(G, V)
Step 1: Convert adjacency matrix to edge list E[]
Step 2: Sort E[] by increasing weight
Step 3: For i = 1 to V
            parent[i] = i
Step 4: mincost = 0
            count = 0
Step 5: For each edge (u, v, w) in E[]
```

```
        If FIND(u) ≠ FIND(v)
          UNION(u, v)
          Print (u, v, w)
          mincost = mincost + w
          count = count + 1
        If count == V - 1
          break
  Step 6: Print mincost
  FIND(x):
     if parent[x] == x
        return x
     return FIND(parent[x])
  UNION(x, y):
     parent[FIND(y)] = FIND(x)
```

Time Complexity:

For the construction of an undirected graph with n vertices and e edges using adjacency list is O(n+e), since for every vertex v in G we need to store all adjacent edges to vertex v.

- In Prim's algorithm to get minimum spanning tree from an undirected graph with n vertices using adjacency matrix is O(n2).
- Using Kruskal's algorithm
  using adjacency matrix = O(n2).
  using adjacency list=O(eloge)

Test Cases:

a) Display the total number of comparisons required to construct the graph in computer memory.
b) Display the results as given in the sample o/p above.
c) Finally conclude on time & time space complexity for the construction of the graph and for generation of minimum spanning tree using Prim's algorithm.

Expected Output: (Results / Visualization)

1. Formatted display (Tree like structure)
2. Appropriate feedback/messages

Inference:

- Prim's and Kruskal's algorithms are greedy techniques used to find the Minimum Spanning Tree (MST) of a weighted, connected graph.
- Prim's algorithm grows the MST from a starting vertex and is suitable for dense graphs, while Kruskal's algorithm builds the MST by selecting the smallest edges and avoiding cycles, making it efficient for sparse graphs.

- Both algorithms ensure a spanning tree with minimum total weight, and the choice between them depends on the graph structure.

Oral questions:

1. Explain the Prim’s algorithm for minimum spanning tree.
2. What are the traversal techniques?
3. What are the graph representation techniques?
4. Why does Prim’s algorithm is faster on denser graphs while Kruskal’s algorithm on sparse graphs?
5. How does Prim’s algorithm work?
6. Does Prim’s algorithm form cycles?
7. What data structure is used in Prim’s algorithm?
8. Time complexity of Prim’s algorithm?
9. Is Prim’s similar to Dijkstra’s algorithm?
10. How does Kruskal’s algorithm work?
11. How does Kruskal’s detect cycles?
12. Time complexity of Kruskal’s algorithm?
13. Can Kruskal’s work on disconnected graphs?
14. Difference between Prim’s and Kruskal’s?
15. Do both algorithms always give the same MST?

Assignment No: 6

Title: Graph- Shortest Path Algorithm

Aim: To represent a city map using a graph data structure (adjacency matrix and adjacency list) where nodes represent landmarks and edges represent distances, and to determine the shortest path from a single source landmark to all other landmarks using Dijkstra's algorithm.

Objectives:

1. To understand graph representation using adjacency matrix and adjacency list.
2. To model real-world city landmarks as nodes and distances as weighted edges.
3. To implement Dijkstra's algorithm for single-source shortest path.
4. To compute the shortest distance from a given source to all other nodes.
5. To analyse the time and space complexity of the implemented algorithm.

CO Mapped: CO2

Problem Statement:

Represent a graph of the city using an adjacency matrix /adjacency list. Nodes should represent the various landmarks and links should represent the distance between them. Find the shortest path using Dijkstra's algorithm from single source to all destinations. Analyse the implemented algorithm for space and time complexity.

Expected Outcome:

1. Successful representation of a city map using graph data structures.
2. Accurate calculation of the shortest distance from the source landmark to all other landmarks.
3. Better understanding of how Dijkstra's algorithm works in real-world applications.
4. Clear comparison between adjacency matrix and adjacency list representations.
5. Insight into algorithm efficiency through complexity analysis.

Pre-requisites:

4. Basic knowledge of graph theory.
5. Understanding of weighted graphs.
6. Familiarity with arrays, lists, and priority queues.
7. Knowledge of greedy algorithms.
8. Basic programming concepts (loops, conditionals, functions).

Suggested Study Material: (Blogs / Videos / Courses / Web Sites / Books / e-Books)

- https://www.geeksforgeeks.org/dsa/dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://www.w3schools.com/dsa/dsa_theory_graphs_shortestpath.php

- https://www.geeksforgeeks.org/dsa/shortest-path-algorithms-a-complete-guide/
- https://www.tutorialspoint.com/data_structures_algorithms/dijkstras_shortest_path_algorithm.htm

Implementation Requirements: (Platform / Software)
Platform: Intellij IDEA
Input: Nodes- 6 to 8, A weighted graph $G(V, E)$with non-negative edge weights A source vertex s

Theory / Procedure / Diagrams:

- Concepts of Graph,
- BST Properties
- BST Operations with example [Insert, Search, Delete, Traversal etc.]
- Graph Applications

Algorithm / Methods / Steps:

➢ **Steps**

1. Initialize an array dist[] for all vertices:
   - Set dist[s] = 0
   - Set dist[v] = ∞ for all other vertices $v \neq s$
2. Create a set visited[] and initialize all values to false.
3. Repeat the following steps until all vertices are visited:
   1. Select the unvisited vertex u with the minimum distance from dist[].
   2. Mark vertex u as visited.
   3. For each adjacent vertex v of u:

If visited[v] == false and
dist[u] + weight(u, v) < dist[v], then:
Update
dist[v] = dist[u] + weight(u, v)

   4. After all vertices are processed, the array dist[] contains the shortest distances from the source to all vertices.
   5. Display the shortest distances.

Test Cases:

Test algorithms on:

1. Simple City Graph
2. Disconnected Graph
3. Single Vertex Graph
4. Fully Connected Graph
5. Zero Weight Edges

Expected Output: (Results / Visualization)

1. The program successfully computes the shortest distance from the given source vertex to all other vertices in the graph using Dijkstra’s algorithm.
2. Format of representation of shortest path in output should be clear and well formatted.

Inference:

The experiment demonstrates that Dijkstra’s algorithm efficiently computes the shortest path from a given source to all reachable destinations in a weighted graph with non-negative edge weights.

Oral questions:

1. What is a graph?
2. What is the difference between adjacency matrix and adjacency list?
3. Why does Dijkstra’s algorithm not work with negative edge weights?
4. What type of algorithm is Dijkstra’s algorithm?
5. What is meant by a single-source shortest path?
6. How is a city map represented as a graph?
7. Which graph representation is more space-efficient and why?
8. What data structure is commonly used to optimize Dijkstra’s algorithm?
9. What happens if the graph is disconnected?
10. State a real-life application of Dijkstra’s algorithm.

Assignment No: 7

Title: KMP for string matching

Aim: To implement the Knuth–Morris–Pratt (KMP) string matching algorithm to efficiently search for a given keyword (pattern) within a large text in a library management system.

Objectives:

1. To understand the limitations of brute-force string matching algorithms.
2. To study the working principle of the KMP algorithm.
3. To preprocess the pattern using the LPS (Longest Prefix Suffix) array.
4. To implement KMP for fast keyword searching in large texts.
5. To improve search efficiency in a text search engine.

CO Mapped: CO3

Problem Statement:

A **library management system** is developing a **text search engine** to help users quickly locate books, research papers, and articles by searching for specific keywords within large texts. Traditional **brute-force string matching algorithms** are inefficient for searching in **large datasets**, leading to slower search results. Implement KMP for string matching in a text search engine.

Expected Outcome:

1. The system efficiently locates all occurrences of a keyword in the given text.
2. Redundant comparisons are avoided during the search process.
3. Faster search results compared to traditional brute-force methods.
4. Successful handling of large text data in library systems.
5. Accurate reporting of pattern positions in the text.

Pre-requisites:

1. Knowledge of strings and arrays.
2. Understanding of basic searching algorithms.
3. Familiarity with time complexity concepts.
4. Understanding of prefix and suffix concepts.

Suggested Study Material: (Blogs / Videos / Courses / Web Sites / Books / e-Books)

- https://www.geeksforgeeks.org/dsa/kmp-algorithm-for-pattern-searching/
- https://takeuforward.org/data-structure/kmp-algorithm
- https://www.tutorialspoint.com/data_structures_algorithms/knuth_morris_pratt_algorithm.htm

Implementation Requirements: (Platform / Software)
Platform: Intellij IDEA
Input: A text string (library content such as books, articles, or research papers). A pattern string (keyword to be searched).

Theory / Procedure / Diagrams:

The Knuth–Morris–Pratt (KMP) algorithm is an efficient string-matching algorithm that avoids unnecessary comparisons by utilizing information gained from previous matches. Unlike brute-force algorithms that recheck matched characters, KMP preprocesses the pattern and constructs an LPS (Longest Prefix Suffix) array.

The LPS array indicates the longest proper prefix of the pattern that is also a suffix. During mismatch, the algorithm uses this array to skip comparisons, making the search process faster and more efficient.

Time Complexity:

- Preprocessing (LPS): O(m)
- Searching: O(n)
  where *n* is the length of the text and *m* is the length of the pattern.

Algorithm / Methods / Steps:

- ➢ Step 1: Construct LPS Array
  1. Initialize LPS array of size *m*.
  2. Set lps[0] = 0.
  3. Compare characters of the pattern to build LPS values.
  - ➢ Step 2: Pattern Searching
  1. Initialize indices i = 0 (text), j = 0 (pattern).
  2. If text[i] == pattern[j], increment both indices.
  3. If j == m, a match is found; record the index and set j = lps[j-1].
  4. If mismatch occurs:
     - o If j != 0, set j = lps[j-1].
     - o Else increment i.
  5. Repeat until the end of the text is reached.

Test Cases:

Test algorithms on:
1. To check whether the keyword exists in the given text.
2. To search for the keyword within the text.
3. To check for multiple occurrences of the keyword in the text.
4. Handles the case where the search pattern is empty.

Expected Output: (Results / Visualization)

The program correctly identifies the presence or absence of the given keyword in the text using the KMP string matching algorithm.

Inference:

The experiment demonstrates that the KMP algorithm provides an efficient and reliable method for keyword searching in large text datasets by avoiding redundant comparisons.

Oral questions:

1. What is the main drawback of brute-force string matching?
2. What does LPS stand for in KMP?
3. Why is KMP more efficient than naive string matching?
4. What is the time complexity of KMP algorithm?
5. Can KMP handle large texts efficiently?
6. What happens when a mismatch occurs in KMP?
7. Is KMP suitable for real-time text searching?
8. Where is KMP used in real-world applications?
9. What is the difference between prefix and suffix?
10. Can KMP be used for multiple pattern searching?