Step 1: Understand Linked Lists

A linked list is a dynamic data structure where each element, called a node, points to the next node in the list. There are two main types of linked lists:

Singly Linked List: In a singly linked list, each node only points to the next node in the list.

Doubly Linked List: In a doubly linked list, each node points to both the previous and next nodes in the list.

Step 2: Setup

Create a Task class with attributes like taskId, taskName, and status:

public class Task {

private int taskId;

private String taskName;

private String status;

public Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

// Getters and setters

public int getTaskId() {

return taskId;

}

public void setTaskId(int taskId) {

this.taskId = taskId;

}

public String getTaskName() {

return taskName;

}

public void setTaskName(String taskName) {

this.taskName = taskName;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

}

Step 3: Implementation

public class TaskLinkedList {

private Node head;

public TaskLinkedList() {

head = null;

}

private class Node {

Task task;

Node next;

public Node(Task task) {

this.task = task;

this.next = null;

}

}

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

public Task searchTask(int taskId) {

Node temp = head;

while (temp != null) {

if (temp.task.getTaskId() == taskId) {

return temp.task;

}

temp = temp.next;

}

return null;

}

public void traverseTasks() {

Node temp = head;

while (temp != null) {

System.out.println(temp.task.getTaskName() + " - " + temp.task.getStatus());

temp = temp.next;

}

}

public void deleteTask(int taskId) {

if (head == null) {

return;

}

if (head.task.getTaskId() == taskId) {

head = head.next;

return;

}

Node temp = head;

while (temp.next != null) {

if (temp.next.task.getTaskId() == taskId) {

temp.next = temp.next.next;

return;

}

temp = temp.next;

}

}

}

Step 4: Analysis

Time Complexity Analysis:

Add Task: O(n) - Adding a task takes linear time, where n is the number of tasks.

Search Task: O(n) - Searching for a task takes linear time, where n is the number of tasks.

Traverse Tasks: O(n) - Traversing tasks takes linear time, where n is the number of tasks.

Delete Task: O(n) - Deleting a task takes linear time, where n is the number of tasks.

Advantages of Linked Lists over Arrays:

Dynamic Size: Linked lists can dynamically resize as elements are added or removed.

Efficient Insertion and Deletion: Linked lists provide efficient insertion and deletion operations, especially when compared to arrays.

Good Cache Locality: Linked lists can provide good cache locality, especially when traversing the list in a linear fashion.

In conclusion, linked lists are a suitable data structure for dynamic task management systems where tasks need to be added, deleted, and traversed efficiently.