Design and Analysis of Algorithms (DAA) Report

# 1. Members

|  |  |  |
| --- | --- | --- |
| Name | Registration Number | Access No |
| Abaho Joy | M23B23/001 | B20228 |
| Isooba Mbeiza Rachel | M23B23/047 | B23293 |
| Rubagumya Alvin | M23B23/012 | B20239 |

# 2. Introduction to the Problem

This task management system aims to provide a solution for managing, sorting, and scheduling. Key components used are task creation, task sorting, task scheduling using Dynamic Programming, task notifications, status updates, and visualizing tasks through Gantt charts.

# 2. DAA Concepts Used

## 2.1 Dynamic Programming for Task Scheduling

Dynamic Programming (DP) was used to find the optimal task schedule based on available time and priority, with academic tasks having more weight than personal ones. This approach was chosen because it breaks down the problem into simpler subproblems, storing solutions to avoid redundant calculations, thus ensuring efficiency.

The task scheduling problem is a variation of the Knapsack Problem. This is seen in the use of dynamic programming, where the state transitions tracks the best possible set of tasks for each time and available capacity.

## 2.2 Binary Search for Task Insertion

**Time Complexity:** O(log n) for each insertion.

Binary Search was used to insert tasks in sorted order based on the starting time. This ensures tasks are inserted at the correct position while maintaining the sorted state of the tasks list.

Since tasks are always kept in sorted order, Binary Search allows for quickly finding the correct insertion index without needing to traverse the entire list.

The use of Binary Search improves the efficiency of insertion, preventing the need to re-sort the list after each addition.

## 2.3 Quick Sort for Sorting Tasks

**Time Complexity:** Best case: O(n log n), Worst case: O(n^2).

Quick Sort was used to sort tasks based on criteria like priority, type, start time, or deadline.

Quick Sort was preferred over other sorting algorithms like Merge Sort because of its average case performance and its ability to sort tasks efficiently, especially when the tasks are randomly distributed.

## 2.4 Heap Operations (Priority Queue)

**Time Complexity:** O(log n) for insertion and deletion.

Heap operations, or priority queues, were used for manage and prioritize events based on their times, such as task start times and deadlines.

The min-heap stored **notifications for task events**. Specifically:

1. **Task start times**.
2. **Task deadlines**.

Each entry in the heap is a tuple structured where:

* event\_time: The datetime for the event (start or deadline).
* event\_type: A string indicating the type of event ("start" or "deadline").
* task: The task object associated with the event.

The heap ensures that events are ordered by their event\_time, with the earliest event at the top.

Without a Priority Queue, tasks would need to be manually sorted each time, making the process more cumbersome and less efficient

# 3. Psuedocode

Create Tasks = [] # Empty list to store tasks

# Function to Create a Task

Function CreateTask():

Input Task Name

Input Task Type # "Academic" or "Personal"

Input Start Date and Time

Input Deadline

Input Priority # Higher numbers = higher priority

Input Duration (in hours)

Input Status # "upcoming", "ongoing", or "completed"

currentTask = {TaskName, TaskType, StartDate, Deadline, Priority, Duration, Status}

Index = BinarySearch(Tasks, currentTask) # Find position to insert task

Insert currentTask into Tasks at Index # Maintain sorted order

# Function for Binary Search

Function BinarySearch(Tasks, currentTask):

Initialize low = 0

Initialize high = length of Tasks - 1

While low <= high:

mid = (low + high) // 2 # Find middle index

If Tasks[mid] < currentTask:

low = mid + 1 # Move to right half

Else:

high = mid - 1 # Move to left half

Return low # Return position to insert task

# Function to Send Notifications

Function Notifications():

CurrentTime = current system time

For each event in Tasks:

If eventTime <= CurrentTime:

If event == "start":

Print("Task started")

TaskStatus = "ongoing"

Else If event == "deadline":

Print("Task reached its deadline")

Else:

Print("Time left till next event")

# Function to Update Task Status

Function UpdateStatus():

CurrentTime = current system time

For each task in Tasks:

If Task Start Time == CurrentTime:

Task Status = "ongoing"

Else If Task Deadline <= CurrentTime:

Output("Have you completed the task? (y/n)")

Input answer

If answer == "y":

Task Status = "completed"

Else If answer == "n":

Task Status = "missed"

Else:

Print("Invalid input")

Function QuickSort(Answer, Task)

If length of Tasks <= 1:

Return Tasks

# Choose the middle Task as the pivot

MiddleIndex = length of Tasks // 2

Pivot = Tasks[MiddleIndex]

# Initialize lists for lower and higher tasks

Lower = []

Higher = []

# Split tasks based on the sorting criteria

If Answer == "priority":

For each task in Tasks excluding Pivot:

If Task priority > Pivot priority:

Add task to Lower

Else:

Add task to Higher

Else If Answer == "type":

For each task in Tasks excluding Pivot:

If Task type > Pivot type:

Add task to Lower

Else:

Add task to Higher

Else If Answer == "start":

For each task in Tasks excluding Pivot:

If Task Start date and time < Pivot Start date and time:

Add task to Lower

Else:

Add task to Higher

Else If Answer == "end":

For each task in Tasks excluding Pivot:

If Task deadline < Pivot deadline:

Add task to Lower

Else:

Add task to Higher

# Recursively sort the lower and higher lists

SortedLower = QuickSort(Lower, Answer)

SortedHigher = QuickSort(Higher, Answer)

# Combine the sorted lists and pivot

Return SortedLower + [Pivot] + SortedHigher

Function SortTasks(Answer):

Tasks = QuickSort(Tasks, Answer)

# Step 2: Display the sorted tasks

For each Task in Tasks:

Output ("Task: " + Task Name +

", Type: " + Task Type +

", Start: " + Task Start date and Time +

", Deadline: " + Task Deadline +

", Priority: " + Task Priority +

", Duration: " + Task Duration + " hours")

Function ScheduleTasks(AvailableTime):

# Step 1: Initialize DP table and selected tasks table

n = Length(Tasks)

DP = Create 2D list of size (n+1) x (AvailableTime+1) with initial value 0

SelectedTasks = Create 2D list of size (n+1) x (AvailableTime+1) with empty sublists

# Step 2: Function to calculate adjusted priority

Function GetPriority(Task):

If Task Type == "academic":

Return Task priority # Academic tasks have full priority

Else:

Return Task priority // 2 # Personal tasks have half priority

# Step 3: Iterate over each task and available time

For i = 1 to n:

For t = 0 to AvailableTime:

Task = Tasks[i - 1] # Current task being considered

# Get adjusted priority for the current task

AdjustedPriority = GetPriority(Task)

# Step 4: Check if the task can be scheduled within the available time

If Task Duration <= t:

# If including this task gives a higher priority, include it

If DP[i - 1][t] < DP[i - 1][t – Task Duration] + AdjustedPriority:

DP[i][t] = DP[i - 1][t – Task Duration] + AdjustedPriority

SelectedTasks[i][t] = SelectedTasks[i - 1][t – Task Duration] + [Task]

Else:

DP[i][t] = DP[i - 1][t]

SelectedTasks[i][t] = SelectedTasks[i - 1][t]

Else:

# If task duration is more than available time, don't include it

DP[i][t] = DP[i - 1][t]

SelectedTasks[i][t] = SelectedTasks[i - 1][t]

# Step 5: Return the selected tasks for the given available time

Return SelectedTasks[n][AvailableTime]

Function to Create Gantt Chart to represent this

Function PrintMenu():

# Display the menu options to the user

Print("\nTask Management System")

Print("1. Add Task")

Print("2. Sort Tasks")

Print("3. Check Notifications")

Print("4. Update Task Statuses")

Print("5. View Gantt Chart")

Print("6. Schedule Tasks")

Print("7. Exit")

Function GetTaskInput():

# Get task details from the user

TaskName = Input("Enter task name: ")

TaskType = Input("Enter task type (academic/personal): ").lower()

TaskStart = Input("Enter task start time (YYYY-MM-DD HH:MM): ")

Deadline = Input("Enter task deadline (YYYY-MM-DD HH:MM): ")

Priority = Input("Enter task priority (higher number is higher priority): ")

Duration = Input("Enter task duration in hours: ")

# Convert input times to datetime format

TaskStart = ConvertToDatetime(TaskStart)

Deadline = ConvertToDatetime(Deadline)

# Create and return the task object

Return Task(TaskName, TaskType, TaskStart, Deadline, Priority, Duration)

Function Main():

While True:

# Display the menu

Call PrintMenu()

Choice = Input("Choose an option (1-7): ")

If Choice == '1':

# Add a new task

Task = Call GetTaskInput()

Call system.AddTaskSorted(Task)

Print("Task '" + Task.TaskName + "' added successfully.")

Else If Choice == '2':

# Sort tasks by chosen criteria

Print("\nSort by:")

Print("1. Priority")

Print("2. Task Type")

Print("3. Start Time")

Print("4. End Time")

SortChoice = Input("Choose an option (1-4): ")

If SortChoice == '1':

Call SortTasks("priority")

Else If SortChoice == '2':

Call SortTasks("type")

Else If SortChoice == '3':

Call SortTasks("start")

Else If SortChoice == '4':

Call SortTasks("end")

Else:

Print("Invalid choice.")

Else If Choice == '3':

# Check for notifications

Print("\nChecking notifications...\n")

Call CheckNotifications()

Else If Choice == '4':

# Update task statuses

Print("\nUpdating task statuses...\n")

Call UpdateStatuses()

Else If Choice == '5':

# View the Gantt chart

Print("\nDisplaying Gantt chart...\n")

Call GanttChart()

Else If Choice == '6':

# Schedule tasks

AvailableTime = Input("Enter the available time in hours: ")

SelectedTasks = Call system.ScheduleTasks(AvailableTime)

Print("\nSelected tasks based on available time:")

For Each Task in SelectedTasks:

Print("Task: " + Task Name + ", Type: " + Task Type + ", Duration: " + Task Duration + " hours, Priority: " + Task Priority)

Else If Choice == '7':

# Exit the program

Print("Exiting the system.")

Break

Else:

Print("Invalid choice, please try again.")