**GROUP MEMBERS**

* Ogwang Andrew
* Puoch Mabor
* Opifudrira Timothy

**1. Project Overview**

The **Personal Scheduling Assistant** is a Python-based application designed to help users efficiently manage and prioritize their daily tasks. The project uses **object-oriented programming** principles and integrates dynamic programming for task optimization, while also providing a Gantt chart visualization to present the task schedule graphically.

**Objectives:**

* Manage tasks with attributes like description, deadline, priority, type, and duration.
* Sort and display tasks based on different criteria (deadline, priority, type).
* Optimize the task schedule to maximize priority within a limited available time.
* Visualize the task schedule using a Gantt chart.

**2. Key Features**

**a. Task Management**

The system allows users to:

* Add tasks with details such as:
  + **Task Description**: Brief explanation of the task.
  + **Deadline**: Date and time by which the task must be completed.
  + **Priority**: Importance level of the task (on a scale of 1 to 10).
  + **Task Type**: Categorize the task as either personal or academic.
  + **Duration**: Estimated time required to complete the task in minutes.

**b. Task Sorting**

Tasks can be sorted by:

* **Deadline** (ascending order).
* **Priority** (descending order, i.e., higher priority tasks come first).
* **Task Type** (personal or academic).

**c. Schedule Optimization**

* Uses **Dynamic Programming (DP)** to select tasks that maximize the total priority while fitting within the available time limit provided by the user.
* The optimization algorithm is based on the **0/1 Knapsack Problem**, where:
  + Task duration acts as the "weight."
  + Task priority acts as the "value."

**d. Gantt Chart Visualization**

* Displays tasks in a graphical format, showing the duration and start time of each task using **Matplotlib**.
* Helps users understand how tasks are distributed over time.

**e. Binary Search**

**How It Works**

1. **Input:**
   * **task\_list: A list of tasks sorted by their deadlines.**
   * **target\_deadline: The deadline you are searching for in the list.**
2. **Initialization:**
   * **low: The starting index of the search range (initially 0).**
   * **high: The ending index of the search range (initially the last index in the list).**
3. **Loop:**
   * **The search continues as long as low is less than or equal to high.**
   * **A middle index, mid, is calculated as the average of low and high.**
4. **Comparison:**
   * **If the deadline of the task at mid matches the target\_deadline, the function returns the task.**
   * **If the target\_deadline is later than the middle task's deadline, the search range is adjusted to the upper half (low = mid + 1).**
   * **Otherwise, the search range is adjusted to the lower half (high = mid - 1).**
5. **End Condition:**
   * **If no match is found, the function returns None.**

**3. System Design**

**a. Class Diagram**

1. **Task Class**
   * Attributes:
     + task\_id: Unique identifier for the task.
     + description: Description of the task.
     + deadline: Deadline for task completion (in datetime format).
     + priority: Priority of the task (integer).
     + task\_type: Type of the task (personal or academic).
     + duration: Duration of the task in minutes.
   * Methods:
     + \_\_lt\_\_(): Compares tasks based on their deadlines.
2. **Scheduler Class**
   * Attributes:
     + tasks: A list of tasks managed as a **min-heap** based on deadlines.
   * Methods:
     + add\_task(task): Adds a task to the scheduler.
     + get\_sorted\_tasks(by): Returns tasks sorted by the specified criterion (deadline, priority, type).
     + optimize\_schedule(total\_minutes): Returns a list of tasks optimized for the available time using dynamic programming.
     + plot\_schedule(): Displays the Gantt chart of tasks.

**4. Implementation**

**a. Dynamic Programming for Task Optimization**

The project implements a **0/1 Knapsack algorithm** to solve the task scheduling problem.

**Steps:**

1. Initialize a 2D DP table dp[i][t], where:
   * i represents the number of tasks considered.
   * t represents the available time in minutes.
2. For each task i:
   * If the task’s duration is less than or equal to the available time t, choose the maximum between:
     + Not including the task: dp[i-1][t]
     + Including the task: dp[i-1][t-task.duration] + task.priority
3. Backtrack through the DP table to identify the tasks selected for the optimized schedule.

**b. Gantt Chart Visualization**

The Gantt chart displays tasks using **Matplotlib**. Each task is represented by a horizontal bar indicating its start time relative to the current time and its duration.

**5. Example Usage**

**Adding Tasks**

text

Copy code

Task Description: Complete math homework

Deadline: 2023-11-30 16:00

Priority: 8

Task Type: academic

Duration: 60 minutes

Task added successfully!

**Viewing Tasks**

text

Copy code

ID: 1, Desc: Complete math homework, Deadline: 2023-11-30 16:00, Priority: 8, Type: academic, Duration: 60 mins

**Optimizing Schedule**

text

Copy code

Enter available time in minutes: 120

Optimized Tasks:

- Complete math homework (Priority: 8)

**6. Evaluation**

**Strengths:**

* **Efficient Task Management:** Provides an intuitive way to add, sort, and display tasks.
* **Optimization:** Uses a dynamic programming approach to maximize task priority within limited time.
* **Visualization:** The Gantt chart provides a clear overview of scheduled tasks.

**Limitations:**

* The current version is **CLI-based**, which may not be user-friendly for all users.
* No task persistence is implemented, meaning tasks are lost when the program is closed.

**7. Future Enhancements**

* **Database Integration:** Store tasks in a database to allow persistence across sessions.
* **Enhanced UI:** Develop a web or mobile interface for a more user-friendly experience.
* **Notification System:** Integrate a reminder system to alert users about upcoming deadlines.

**8. Conclusion**

The **Personal Scheduling Assistant** is a useful tool for managing and optimizing daily tasks. With a clear structure and efficient scheduling algorithms, it helps users prioritize important tasks and visualize their schedules effectively. The project demonstrates the integration of core programming concepts, algorithms, and data visualization in a real-world application.

**9. Pseudocode**

START

- Initialize an empty list for tasks.

LOOP:

    User can choose to:

        1. Add a task

        2. Sort tasks by deadline, priority, or type

        3. Search for a task by deadline

        4. View Gantt chart

WHEN adding a task:

    - Get task details from the user.

    - Create a task with description, deadline, priority, type, and duration.

    - Insert the task into the list (keep the list sorted by deadline).

WHEN sorting tasks:

    - Show the list sorted by deadline, priority, or type.

WHEN searching for a task:

    - Get the search deadline from the user.

    - Use binary search to find the task by deadline.

    - Show the task details or an error if not found.

WHEN showing Gantt chart:

    - Display a chart showing all tasks as bars, with the position and length based on their deadline and duration.

END