GROUP MEMBERS

- ITUNGO AGABA
- NKANGI MOSES
- ASINGURA K PHILIP

Technical Documentation: Personal Scheduling Assistant

Overview

The Personal Scheduling Assistant is a Python-based program designed to manage and optimize daily activities. The system helps track deadlines, prioritize tasks, and send reminders for upcoming or missed tasks. The assistant utilizes sorting and searching techniques to handle the prioritization and scheduling of tasks efficiently. Additionally, it provides an algorithmic solution to dynamic scheduling problems using **Dynamic Programming (DP)**.

Features

1. **Task Management**: Add, view, and schedule tasks.
2. **Sorting**: Tasks are sorted by deadlines and priority.
3. **Searching**: Ability to search for tasks by name.
4. **Task Scheduling**: Uses Dynamic Programming (DP) to optimize task scheduling.
5. **Reminder System**: Notifies users of upcoming or missed tasks.
6. **Task Density Analysis**: Provides analysis of task density over specified time intervals.
7. **Gantt Chart**: Visual representation of task scheduling.
8. **User Interface**: Command-line based interface for task input and interaction.

Modules

1. Task Class

The `Task` class is used to represent individual tasks in the scheduling system. It contains the following attributes:

- `name`: The name of the task (string).
- `task_type`: The type of task, either `"personal"` or `"academic"` (string).

- `deadline`: The deadline of the task, represented as a `datetime` object.
- `priority`: The priority level of the task, with lower values representing higher priority (integer).
- `duration`: The duration required to complete the task in minutes (integer).

Pseudo Code: Task Class

```
Class Task:
    Attributes:
        name, task_type, deadline, priority, duration

    Constructor:
        Initialize the task with name, task_type, deadline, priority,
duration
```

---

2. TaskManager Class

The `TaskManager` class manages a collection of tasks and implements various functionalities like adding tasks, sorting tasks, and scheduling them.

Key Methods

- **`add_task(task)`**: Adds a task to the priority queue.
- **`get_upcoming_tasks()`**: Retrieves tasks sorted by their deadlines.
- **`track_reminders()`**: Tracks tasks that are upcoming or missed.
- **`schedule_tasks_dp()`**: Uses Dynamic Programming to schedule tasks optimally.
- **`sort_tasks_by_priority()`**: Sorts tasks by their priority levels.

Sorting and Searching Algorithms

1. Sorting Tasks by Deadline (Using Heap and List Sorting)

The tasks in the `TaskManager` are stored in a priority queue (min-heap) based on priority, and they can be sorted by deadline for optimal scheduling. Sorting by deadline ensures that tasks that are due earlier are handled first.

- **Sorting by Deadline**:
    - In the `get_upcoming_tasks()` method, tasks are sorted by their deadlines to display tasks in chronological order.

Pseudo Code: Sorting Tasks by Deadline

```
Method get_upcoming_tasks():
    Sort tasks based on their deadline attribute
```

```
    Return the sorted tasks
```

Sorting Implementation:

- Tasks are stored in a priority queue (`heapq.heappush()`), and when retrieved, they are sorted by their deadline.

---

2. Dynamic Programming Scheduling (Task Scheduling Optimization)

To optimize the scheduling of tasks, the system uses **Dynamic Programming (DP)**. The goal is to maximize the number of tasks that can be scheduled within their respective deadlines.

The method `schedule_tasks_dp()` implements the DP approach, where:

- Tasks are sorted by deadline.
- The DP array `dp[i]` holds the maximum number of tasks that can be scheduled up to task `i`.
- The algorithm iterates through each task, checking if it can be scheduled based on the previous scheduled tasks and the available time.

Pseudo Code: Dynamic Programming Scheduling

```
Method schedule_tasks_dp():
    Sort tasks by deadline

    Initialize DP array dp[0..n] and set dp[i] to 0 (no tasks scheduled
initially)
    Initialize a schedule array to track the scheduled tasks

    For each task i from 1 to n:
        For each previous task j from i-1 to 1:
            If task i can be scheduled (i.e., task j's deadline <= task i's
deadline - task i's duration):
                Update dp[i] = max(dp[i-1], dp[j] + 1)

    Create a scheduled task list based on the DP array and return it
```

Dynamic Programming Analysis:

- **Time Complexity**: $O(n^2)$ where `n` is the number of tasks, as each task is compared with all prior tasks.
- **Space Complexity**: O(n) for the DP array and schedule list.

---

3. Task Density Analysis (Density Calculation)

The `analyze_task_density()` method calculates task density over specific time intervals. It counts how many tasks fall within each time window and generates a density chart.

Pseudo Code: Task Density Analysis

```
Method analyze_task_density():
    Initialize an empty dictionary intervals
    Calculate the earliest and latest task deadlines
    Define the time interval (default is 60 minutes)

    For each time window (starting from the earliest task deadline):
        Count tasks that fall within this window
        Store the count in the intervals dictionary

    Return the intervals dictionary
```

Task Density Calculation:

- Tasks are iterated through and counted for each time interval, generating a density distribution.

Algorithms in Detail

1. Sorting Algorithms

Tasks are sorted by their deadlines in the following scenarios:

- **When displaying upcoming tasks**: Tasks are retrieved from the priority queue and sorted by their deadlines to provide the user with a chronological list.
- **When scheduling tasks dynamically**: Tasks are sorted by deadlines before applying the Dynamic Programming algorithm to optimize the schedule.

Sorting by Deadline (Min-Heap Sorting)

```
heappush(self.tasks, (task.priority, task.deadline, task))  # Add task to
heap
```

In this implementation, the `heapq.heappush()` method is used to insert tasks into the priority queue. Although tasks are initially sorted by priority, they are retrieved and sorted by their deadline in the `get_upcoming_tasks()` method.

2. Searching for Tasks by Name

Searching for a task by name is done via a linear search through the list of tasks stored in the `TaskManager`.

```
Method search_task_by_name(task_name):
    For each task in tasks:
        If task.name matches task_name:
            Return the task
```

User Interface (Menu)

The interface is driven by a command-line menu that allows the user to interact with the system. The main options are:

1. **Add Task**: Allows the user to input task details (name, type, deadline, priority, and duration).
2. **View Upcoming Tasks**: Displays tasks sorted by their deadlines.
3. **Schedule Tasks**: Uses Dynamic Programming to optimize and schedule tasks.
4. **Analyze Task Density**: Analyzes the density of tasks over time intervals.
5. **Track Reminders**: Provides reminders for upcoming or missed tasks.
6. **Exit**: Terminates the program.

Conclusion

This Personal Scheduling Assistant leverages sorting and searching algorithms to efficiently manage tasks, prioritize them based on urgency, and ensure optimal task scheduling. It uses **Dynamic Programming** to solve scheduling problems and provides useful features such as reminders and task density analysis. The assistant can be easily extended to support more complex scheduling features and integrate with other tools or systems.