FACULTY OF ENGINEERING, DESIGN AND TECHNOLOGY

DEPARTMENT OF COMPUTING AND TECHNOLOGY

ADVENT 2024 SEMESTER OOP COURSEWORK PROJECT REPORT

PROGRAM: BSCS, BSDS 2:1

COURSE: DAA

COURSE LECTURER: HABARE WABWIRE

PROJECT TITLE: *PERSONAL SCHEDULING ASSISTANT*

Submitted by

S/N	Reg Number	Name
1.	S23B23/085	ODONGKARA OSCAR
2.	S23B23/047	OBBA MARK CALVIN
3.	S23B23/082	NANKYA ZAHARAH

Date Submitted: 27TH NOVEMBER 2024

---

1. Introduction

The **Personal Scheduling Assistant** is a Python-based application designed to assist users in managing their tasks efficiently. It provides functionalities for task creation, optimization, and visualization, enabling better time management. The tool features a graphical user interface (GUI) for ease of use and leverages dynamic programming for task scheduling optimization.

---

2. System Overview

The application uses:

- **Programming Language**: Python
- **Core Libraries**:
  - tkinter for GUI design.
  - matplotlib for data visualization.
  - datetime for time handling.
  - json for task persistence.
- **Design Paradigm**: Object-Oriented Programming (OOP) with components for tasks, scheduling logic, and user interface.

---

3. Key Features

3.1 Task Management

- Add, delete, and view tasks.
- Tasks include attributes: title, type (e.g., Academic, Work), priority, start time, and end time.

3.2 Scheduling Optimization

- Tasks are scheduled to maximize priority while avoiding time conflicts using dynamic programming.

3.3 Data Persistence

- Tasks are saved to a JSON file (tasks.json) and reloaded upon application restart.

3.4 Visualization

- **Gantt Chart**: Displays tasks in a timeline view.
- **Busy Slots Analysis**: Presents load distribution using bar charts.

3.5 Notifications

- Sends reminders for tasks due within 24 hours.

---

4. Implementation Details

4.1 Task Model

A Task class encapsulates attributes like title, type, priority, start time, and end time. It provides methods for serialization (conversion to dictionary) to support JSON storage.

4.2 Scheduling Logic

The **SchedulingAssistant** class handles:

- Task management (addition, deletion, sorting).
- Task optimization using dynamic programming.
- Data persistence through file I/O.

4.3 User Interface

The **SchedulerGUI** class creates the GUI with the following components:

- Task list displayed in a tree view.
- Input fields for adding tasks.
- Buttons for task analysis and visualization.

---

5. Algorithmic Design

5.1 Task Optimization Algorithm

Objective: Maximize the total priority score of non-overlapping tasks.

- **Input**: A list of tasks sorted by end time.
- **Method**:

1. Use dynamic programming to calculate the maximum achievable priority.
2. Apply binary search to find the latest non-overlapping task for efficient computation.

- **Complexity**:
  - Sorting tasks: O(nlog⁡n)O(n \log n)
  - Dynamic programming: O(nlog⁡n)O(n \log n) (due to binary search).

5.2 Merge Sort for Task Ordering

Tasks are sorted by priority and start time using a custom merge sort implementation for stability and efficiency.

---

6. User Interface Design

The GUI is implemented with **tkinter**:

- **TreeView Widget**: Displays tasks with attributes like title, type, priority, and status.
- **Input Forms**: For task creation, using dropdowns and text fields.
- **Buttons**:
  - Add Task
  - Delete Task
  - Generate Gantt Chart
  - Send Reminders
  - Analyze Busy Slots
  - Maximize Tasks

Charts and graphs are rendered using **matplotlib** and displayed as interactive plots.

---

7. Testing and Validation

7.1 Unit Testing

Key functionalities tested:

- Task addition and deletion.
- Correctness of the optimization algorithm.
- File I/O operations for saving and loading tasks.

7.2 GUI Testing

- Verified correct rendering of widgets and event handling.
- Checked integration with backend logic for task creation and visualization.

7.3 Edge Cases

- Adding tasks with overlapping times.
- Handling invalid input formats for time fields.
- Loading corrupted or empty JSON files.

---

8. Future Enhancements

1. **Task Overlap Validation**:
    - Add checks to prevent overlapping tasks during creation.
2. **Task Filtering and Search**:
    - Allow users to search tasks by type, priority, or date.
3. **Database Integration**:
    - Replace JSON storage with a relational database for scalability.
4. **Recurring Tasks**:
    - Add support for periodic tasks (e.g., weekly meetings).
5. **Theming**:
    - Introduce light/dark mode options for better usability.

---

9. Conclusion

The **Personal Scheduling Assistant** is a functional tool for task management and scheduling. By combining a robust backend with an intuitive GUI, it offers

users a seamless experience for managing their schedules. Its modular design ensures that new features can be easily integrated, making it a promising foundation for further development.

Appendix: Prerequisites

- **Python Version**: 3.7 or higher
- **Dependencies**:
    - matplotlib
    - tkinter (bundled with Python)

To run the application, save the code in a Python script and execute it in a terminal using:

python scheduling_assistant.py

PSEUDOCODE

Class Task:

Attributes:

- title: string

- task_type: TaskType (enum)

- priority: TaskPriority (enum)

- start_time: datetime

- end_time: datetime

- completed: boolean

- reminded: boolean

Methods:

- to_dict():

Return task attributes as a dictionary

Class SchedulingAssistant:

Attributes:

- tasks: list of Task objects

Methods:

- __init__():

Initialize empty task list

Load tasks from "tasks.json" if exists

- add_task(task):

Add a new task to the list

Sort tasks using merge_sort()

Save tasks to "tasks.json"

- save_tasks():

Write all tasks to "tasks.json" as JSON

- load_tasks():

Read tasks from "tasks.json" and populate the list

- maximize_tasks():

Find the optimal non-overlapping tasks with maximum priority

Return list of selected tasks

- display_maximized_tasks():

Show a popup with the list of optimal tasks

- delete_task(title):

Remove a task from the list by title

Save updated tasks

- send_reminders():

Notify users of tasks due within 24 hours

Mark notified tasks as reminded

Save updated tasks

- get_busy_slots():

Return a list of busy slots with start, end, and load (hours)

- generate_gantt_chart():

Display a Gantt chart of all tasks

- merge_sort(tasks):

Sort tasks by priority and start time using merge sort

- merge(left, right):

Merge two sorted lists into one

- compare_tasks(task1, task2):

Compare tasks by priority, then by start time

Class SchedulingAssistant:

Attributes:

- tasks: list of Task objects

Methods:

- __init__():

Initialize empty task list

Load tasks from "tasks.json" if exists

- add_task(task):

Add a new task to the list

Sort tasks using merge_sort()

Save tasks to "tasks.json"

- save_tasks():

Write all tasks to "tasks.json" as JSON

- load_tasks():

Read tasks from "tasks.json" and populate the list

- maximize_tasks():

Find the optimal non-overlapping tasks with maximum priority

Return list of selected tasks

- display_maximized_tasks():

Show a popup with the list of optimal tasks

- delete_task(title):

Remove a task from the list by title

Save updated tasks

- send_reminders():

Notify users of tasks due within 24 hours

Mark notified tasks as reminded

Save updated tasks

- get_busy_slots():

Return a list of busy slots with start, end, and load (hours)

- generate_gantt_chart():

Display a Gantt chart of all tasks

- merge_sort(tasks):

Sort tasks by priority and start time using merge sort

- merge(left, right):

Merge two sorted lists into one

- compare_tasks(task1, task2):

Compare tasks by priority, then by start time

MAIN FUNCTION

```
Function main():
    Create root window
    Instantiate SchedulerGUI with root
    Run Tkinter main loop
```

MERGE SORT

```
Function merge_sort(tasks):
    If tasks length <= 1:
        Return tasks
    Split tasks into left_half and right_half
    Recursively call merge_sort on left_half and right_half
    Return merge(left_half, right_half)

Function merge(left, right):
    Initialize empty sorted_tasks list
    While left and right are not empty:
        Compare the first elements of left and right using compare_tasks()
        Append the smaller element to sorted_tasks
    Append remaining elements of left and right to sorted_tasks
    Return sorted_tasks
```

MAXIMIZE TASKS

```
Function maximize_tasks():
    Sort tasks by end_time
    Initialize dp array to store maximum priority
    Initialize p array for latest non-overlapping tasks

    For each task j:
        Perform binary search to find p[j], the latest non-overlapping task

    Calculate dp[j] as:
        max(dp[j-1], task_priority[j] + dp[p[j]+1])

    Trace back from dp to find selected tasks
    Return selected_tasks
```

GANTT CHART GENERATION

```
Function generate_gantt_chart():
    If no tasks:
        Show message "No tasks to display"
        Return
    For each task:
        Calculate duration and assign color based on priority
        Plot horizontal bar representing task
    Display the chart with formatted axis and labels
```