GROUP MEMEBERS

YAWE ARTHUR SHALOM M23B23/016

NABASA ISAAC M23B23/043

MWEBEMBEZI NICOLE M23B23/026

Project Overview

The Personal Scheduling Assistant is designed to help users manage their schedules effectively by providing functionality to add, sort, and manage tasks. It also supports reminders, task prioritization, Gantt chart visualization, and an analysis of task density.

System Components

1. **Task Management**: Add, sort, and search tasks.
2. **Reminder System**: Schedule and check reminders.
3. **Deadline Monitoring**: Identify missed and upcoming tasks.
4. **Task Optimization**: Prioritize non-overlapping tasks using dynamic programming.
5. **Visualization**: Display tasks in a Gantt chart for clarity.
6. **Density Analysis**: Analyze busy slots based on task density.

Core Functionalities

1.Task Management

Purpose: To add, display, and sort tasks based on criteria like deadline, priority, or type.

Pseudo-code:

FUNCTION add_task(start_time, end_time, task_type, description, priority):

VALIDATE inputs (start_time: datetime, end_time: datetime, task_type: str, description: str, priority: int)

INSERT task into sorted list using bisect.insort by start_time

CALL schedule_reminder(end_time, description)

FUNCTION display_tasks():

IF tasks list is empty: PRINT "No tasks available"

ELSE: FOR task in tasks: PRINT task details

FUNCTION sort_tasks(key):

IF key == 'deadline': SORT tasks by task[1] (end time)

ELSE IF key == 'priority': SORT tasks by -task[4] (descending priority)

ELSE IF key == 'type': SORT tasks by task[2] (type) PRINT sorted tasks

2. Reminder System

Purpose: To notify users about tasks an hour before their deadline.

Pseudo-code:

FUNCTION schedule_reminder(deadline, description):

VALIDATE inputs (deadline: datetime, description: str)

CALCULATE reminder_time = deadline - 1 hour

APPEND (reminder_time, deadline, description) to reminders

FUNCTION check_reminders():

SET current_time = datetime.now()

FILTER reminders where reminder_time >= current_time

PRINT upcoming reminders OR "No upcoming reminders"

3. Deadline Monitoring

Purpose: To identify tasks with missed deadlines and highlight the next five upcoming tasks.

Pseudo-code:

FUNCTION check_deadlines():

SET current_time = datetime.now()

FILTER missed_tasks = tasks WHERE task[1] < current_time

FILTER upcoming_tasks = tasks WHERE task[1] >= current_time

PRINT missed_tasks

PRINT first 5 upcoming_tasks

4. Task Optimization

Purpose: Use dynamic programming to maximize non-overlapping tasks.

Pseudo-code:

FUNCTION prioritize_tasks():

SORT tasks by task[1] (end time)

INITIALIZE dp array with size n (number of tasks)

INITIALIZE dp_schedule array for storing optimal schedules

FOR each task in tasks:

CALCULATE incl (include task) and find non-overlapping tasks

CALCULATE excl (exclude task)

SELECT max(incl, excl) and update dp and dp_schedule

PRINT optimal schedule

5. Visualization

Purpose: Display tasks using a Gantt chart grouped into categories like personal and academic.

Pseudo-code:

FUNCTION plot_gantt_chart():

FILTER tasks into personal_tasks and academic_tasks

PLOT each task as a horizontal bar on the Gantt chart

ANNOTATE each bar with task description

DISPLAY chart

6. Density Analysis

Purpose: Analyze task density in specific time intervals.

Pseudo-code:

FUNCTION analyze_busy_slots(interval_minutes):

SET start_of_day = midnight of current date

GENERATE time intervals of size interval_minutes

COUNT number of tasks overlapping each interval

SORT intervals by task count (descending)

PRINT intervals and task counts

7. Task Searching

Purpose: Search for tasks using a keyword in the description.

Pseudo-code:

FUNCTION search_task(keyword):

FILTER results = tasks WHERE keyword.lower() in task[3].lower()

PRINT matching tasks OR "No tasks found"

Dynamic Programming for Task Prioritization

Approach:

1. Sort tasks by their end times to ensure earlier deadlines are evaluated first.
2. Iterate through the tasks:

   Include the current task and add non-overlapping tasks before it.

   Exclude the current task and use the previous maximum.
3. Use a dp array to store the maximum number of tasks that can be scheduled up to each point.

Gantt Chart Visualization

1. Separate tasks into categories.
2. Use matplotlib to plot tasks as horizontal bars.
3. Annotate each bar with the task description for clarity.

Technologies Used

- Python Libraries: bisect, datetime, matplotlib, seaborn.
- Algorithm: Dynamic Programming, Merge Sort.