UGANDA CHRISTIAN UNIVERSITY

A Centre of Excellence in the Heart of Africa

FACULTY OF ENGINEERING, DESIGN AND TECHNOLOGY
DEPARTMENT OF COMPUTING AND TECHNOLOGY
ADVENT 2024 SEMESTER DAA COURSEWORK TECHNICAL DESIGN DOCUMENT

PROGRAM: BSCS

COURSE: Design and Analysis of Algorithms

COURSE LECTURER: Habare John

PROJECT TITLE:

Personal Scheduling Assistant in Python

Submitted by:

NAME	REGISTRATION NUMBER	ACCESS NUMBER
MOKILI PROMISE PIERRE	M23B23/032	B20848
BUWEMBO DAVID DENZEL	S23B23/074	B25565
LUFENE MARK TRAVIS	S23B23/032	B24263

Table of Contents

Personal Scheduling Assistant

1. Overview

The Personal Scheduling Assistant is a Python-based application designed to help users organize tasks efficiently using dynamic programming techniques, sorting algorithms, and a visual representation of tasks using a Gantt chart. Built using **Tkinter** for the GUI and **Matplotlib** for task visualization, this project demonstrates the application of algorithm design principles.

2. System Architecture

2.1 Components

- **Task Class**: Represents individual tasks with attributes like name, category, priority, deadline, and duration.
- **Scheduler Class**: Handles task management operations such as adding, removing, and visualizing tasks.
- **SchedulerApp Class (GUI)**: Implements the user interface for interacting with the Scheduler.
- **Matplotlib Integration**: Generates a Gantt chart for tasks.

2.2 High-Level Workflow

1. User enters task details via the GUI.
2. The **Task** object is created and stored in the **Scheduler** class.
3. Tasks are sorted and visualized based on deadlines using optimized sorting algorithms (e.g., merge sort).
4. A Gantt chart displays task schedules visually.

3. Features and Functional Requirements

3.1 Features

- Add, remove, and view tasks categorized as **Personal** or **Academic**.

- Assign priorities and deadlines to tasks.
- Generate Gantt charts to display scheduled tasks.
- Sort tasks dynamically based on priority and deadlines.

3.2 Functional Requirements

- Input validation for task attributes.
- Deadlines must follow the format YYYY-MM-DD HH:MM.
- Efficient scheduling using dynamic programming and optimized sorting.
- Handle conflicts where tasks overlap or deadlines are missed.

4. Algorithm Design

4.1 Sorting Tasks by Deadline (Quick sort)

- **Purpose**: To ensure tasks are scheduled in chronological order.
- **Complexity**: O(n^2)

.

Pseudocode:

```
function quick_sort(tasks):
    if length(tasks) ≤ 1:
        return tasks
    pivot = tasks[0]
    left = [task for task in tasks[1:] if task.deadline < pivot.deadline]
    right = [task for task in tasks[1:] if task.deadline ≥ pivot.deadline]
    return quick_sort(left) + [pivot] + quick_sort(right)
```

4.2 Dynamic Programming for Task Scheduling

- **Purpose**: Optimize task scheduling to minimize deadline conflicts.
- **Approach**: Use dynamic programming to calculate the maximum number of tasks that can be scheduled within their deadlines.

Pseudocode:

```
function schedule_tasks(tasks):
```

```
sort tasks by deadline
dp = [0] * (n + 1)
for i from 1 to n:
    for j from i - 1 to 0:
        if tasks[i].deadline >= tasks[j].deadline + tasks[j].duration:
            dp[i] = max(dp[i], dp[j] + 1)
return dp[n]
```

5. Technical Design

5.1 Task Class

Attributes:

- name: Task name.
- category: Task type (Personal/Academic).
- priority: Task importance (1-3).
- deadline: Task deadline as a datetime object.
- duration: Task duration in hours.

Pseudocode:

```
class Task:
    attributes: name, category, priority, deadline, duration
    function __init__(name, category, priority, deadline, duration):
        initialize attributes
```

5.2 Scheduler Class

Methods:

- add_task(task): Adds a task to the list.
- remove_task(task_name): Removes a task by name.
- display_gantt_chart(): Visualizes tasks using Matplotlib.

How the application works:

1. **Initialize Application**:

  - A Scheduler manages the task list and operations (add, remove, sort, optimize).
  - SchedulerApp provides the GUI interface for user interaction.
2. **Add Task**:
   - User inputs task details.
   - Task is added to the scheduler, and the list is displayed in sorted order.
3. **Remove Task**:
   - User selects a task from the displayed list.
   - Scheduler removes it using binary search for efficiency.
4. **Display Gantt Chart**:
   - Tasks are shown visually on a timeline, sorted by deadlines.
5. **Optimal Schedule**:
   - Scheduler computes the maximum number of non-overlapping tasks using dynamic programming.
   - Displays these tasks to the user.
6. **Sort and Search**:
   - Quick sort ensures tasks are ordered efficiently.
   - Binary search allows fast task removal by name.

Pseudocode:

```
class Scheduler:
  attributes: tasks = []


  function add_task(task):
    append task to tasks list


  function remove_task(task_name):
    filter tasks list to exclude task_name


  function display_gantt_chart():
    sort tasks by deadline using mergeSort
    generate Gantt chart with task durations
```

5.3 SchedulerApp Class

Methods:

- add_task(): Collects inputs, validates them, and adds tasks.
- remove_task(): Removes selected task from the scheduler.
- refresh_task_list(): Updates the GUI task list display.
- show_gantt_chart(): Displays the Gantt chart.

Pseudocode:

```
Define class Task:
  Initialize Task with attributes: name, category, priority, deadline, duration
  Define a string representation method for debugging
Define class Scheduler:
  Initialize tasks as an empty list

  Define add_task(task):
    Append task to tasks
    Sort tasks using quick_sort by deadline

  Define quick_sort(arr, key):
    If arr has 1 or fewer elements:
      Return arr
    Select pivot as the middle element based on key
    Divide arr into:
      left: elements with key < pivot
      middle: elements with key == pivot
      right: elements with key > pivot
    Recursively sort left and right, then return concatenated result

  Define binary_search(name):
    Initialize low = 0, high = len(tasks) - 1
    While low <= high:
      Set mid = (low + high) // 2
      If tasks[mid].name == name:
        Return mid
      Else if tasks[mid].name < name:
            Update low = mid + 1
        Else:
            Update high = mid - 1
    Return -1 (name not found)

Define remove_task(task_name):
    Use binary_search to find index of task_name
    If index found:
        Remove task at index

Define max_non_overlapping_tasks():
    Sort tasks by deadline
    Initialize dp array of size n with 0
    Initialize prev array to track task dependencies
    Set dp[0] = 1
    For each task i from 1 to n-1:
        Set include = 1
        For each task j before i (in reverse):
            If task j's deadline + duration <= task i's deadline:
                Update include += dp[j]
                Store j in prev[i]
                Break
        Set dp[i] = max(dp[i-1], include)
    Trace back from dp to find selected tasks
    Return list of selected tasks

Define get_task_list():
    Return tasks

Define display_gantt_chart():
    Sort tasks by deadline
    Initialize a Gantt chart with matplotlib
    For each task:
        Calculate start and end times
        Add task as a bar in the chart
    Display chart
Define class SchedulerApp:
  Initialize with root (Tkinter root) and scheduler (Scheduler object)

  Create input fields:
    Task name (Entry)
    Category (Combobox with ["Personal", "Academic"])
    Priority (Combobox with [1, 2, 3])
    Deadline (Entry, format: "YYYY-MM-DD HH:MM")
    Duration (Entry, in hours)

  Create buttons:
    Add Task (calls add_task)
    Remove Task (calls remove_task)
    Show Gantt Chart (calls show_gantt_chart)
    Optimal Schedule (calls show_optimal_schedule)

  Create Task List (Listbox to display tasks)

  Define add_task():
    Get input values from fields
    Validate input values (correct format, all fields filled)
    If validation fails:
      Show error message
    Else:
      Create a Task object
      Add it to scheduler
      Refresh task list
      Show success message

  Define remove_task():
    Get selected task from Task List
    If task selected:
        Extract task name
        Remove task from scheduler
        Refresh task list
      Show success message
    Else:
      Show error message

  Define refresh_task_list():
    Clear Task List
    For each task in scheduler:
      Add task details to Task List

  Define show_gantt_chart():
    If tasks are empty:
      Show error message
    Else:
      Call scheduler.display_gantt_chart()

  Define show_optimal_schedule():
    Get optimal tasks using scheduler.max_non_overlapping_tasks()
    If no tasks found:
      Show info message
    Else:
      Show tasks in an info message

Run SchedulerApp:
  Create Scheduler object
  Create Tkinter root
  Initialize SchedulerApp with root and scheduler
  Start Tkinter event loop
```

6. Gantt Chart Visualization

The display_gantt_chart method uses Matplotlib to create horizontal bar plots for tasks, showing their start and end times on a timeline. Tasks are labeled by name for clarity.