CPSC 335: Spring 2025 — Semester Project Instructions

Dr. Hassan Shah

Overview

Each group of 3 students will work on **one of the three application-driven semester projects** listed below. These projects are designed to consolidate your understanding of graph algorithms, string matching, dynamic programming, greedy methods, and sorting/searching techniques.

Timeline:
Project duration: **2 weeks**
Final presentation and submission (PPT): In-class on Week 14
Total Points: 50

Submission Guidelines

- Prepare a well-structured **PowerPoint presentation (PPT)**.
- Demonstrate all core algorithms and application components.
- Present in class: each group gets 7–10 minutes.
- Submit PPT file via Canvas before the deadline.

Project Choices (Pick ONE)

1. Smart Campus Navigation and Task Scheduler

Objective: Develop a system to assist CSUF students in navigating the campus efficiently and managing their daily tasks.

Key Features:

- **Graph Representation:** Model the campus as a graph where buildings are nodes and pathways are edges. Use adjacency lists or matrices.
- **Shortest Path Algorithm:** Implement Dijkstra's algorithm to find the shortest path between two buildings. Refer to `dijkstra_algorithm.py`.

- **Minimum Spanning Tree (MST):** Use Prim's or Kruskal's algorithm to determine optimal maintenance routes covering all buildings. See `prim_algorithm.py` and `kruskal_algorithm.py`.
- **Task Scheduling:** Apply the Activity Selection Problem (greedy approach) to schedule tasks without overlaps. Refer to `activity_selection.py`.
- **String Matching:** Implement KMP or Boyer-Moore algorithms to search for building names or room numbers. See `kmp_algorithm.py` and `boyer_moore.py`.
- **Sorting:** Use Merge Sort or Quick Sort to organize tasks by priority or time. Refer to `merge_sort.py` and `quick_sort.py`.

Hints:

- Start by mapping out a simplified version of the campus with 5–10 buildings.
- Create sample data for student schedules and tasks.
- Visualize the graph using libraries like `networkx` and `matplotlib`.
- Ensure modular code for each algorithm to facilitate testing and integration.

Expected Outcomes:

- A user interface (CLI or GUI) allowing input of current location and desired destination, displaying the shortest path.
- A scheduler that inputs tasks with start and end times and outputs an optimized, non-overlapping schedule.
- Search functionality to find specific buildings or rooms.
- Sorted list of tasks based on user-defined criteria.

2. CSUF Document Scanner & Pattern Extractor

Objective: Create a tool to scan student documents, detect plagiarism, and extract key information.

Key Features:

- **String Matching:** Use Rabin-Karp and KMP algorithms to detect duplicate phrases or plagiarized content. Refer to `rabin_karp.py` and `kmp_algorithm.py`.
- **Naive Search:** Implement for real-time search in documents. See `naive_search.py`.
- **Compression:** Apply Huffman Coding to compress extracted data. Refer to `huffman_encoding.py`.
- **Graph Traversal:** Model citations or references as a graph and use BFS/DFS to analyze connections. See `bfs.py` and `dfs.py`.

- **Sorting:** Organize files by author, title, or date using Merge Sort or Counting Sort. Refer to `merge_sort.py` and `counting_sort.py`.
- **Optimization:** Use greedy or dynamic programming approaches to prioritize scanning based on relevance.

Hints:

- Utilize plain text files for document input.
- Generate sample documents with intentional overlaps for testing plagiarism detection.
- Visualize citation graphs to understand reference structures.
- Modularize code for each functionality to allow independent testing.

Expected Outcomes:

- A system that inputs documents and outputs detected plagiarized sections with references.
- Compressed versions of documents using Huffman Coding.
- Visual representation of citation networks.
- Sorted document lists based on selected criteria.

3. City Emergency Response and Resource Manager

Objective: Design a system to manage emergency responses in a city, optimizing routes and resource allocation.

Key Features:

- **Graph Modeling:** Represent city zones as a graph with nodes and weighted edges indicating distances or travel times.
- **Shortest Path Algorithm:** Use Dijkstra's algorithm to route emergency vehicles. Refer to `Dijkstra_Test.py`.
- **MST Planning:** Use Prim's or Kruskal's to simulate infrastructure building or emergency coverage zones. Refer to `Prims_algo.py` or `Kruskal_test.py`.
- **Incident Scheduling:** Use the Activity Selection or 0/1 Knapsack Problem to assign responders to non-overlapping incidents. Refer to `Activity_Selection_03072025.py` and `Knapsack_Problem_03082025.py`.
- **Incident Log Search:** Use Rabin-Karp or KMP to search logs or messages for keywords like "fire", "injury", etc. Refer to `RabinKarp_test.py` and `KMP_test.py`.
- **Sorting Logs:** Sort incidents by priority or time using Merge Sort or Quick Sort. Refer to `MergeSort_02072025.py` or `QuickSort_02212025.py`.

- **Optional Compression:** Use Huffman Coding to compress emergency logs. See `Huffman_Code_03082025.py`.

Hints:

- Design a city map with 8–12 regions, and define edge weights based on estimated travel times.
- Create at least 5–10 simulated emergencies with different types, times, and priorities.
- Assign resources like fire trucks, ambulances, etc., and show optimized allocation.
- Visualize routing with console maps or use libraries like `matplotlib` or `networkx`.

Expected Outcomes:

- Visual output of the city map with routing from one zone to another.
- Console/GUI interface that shows real-time incident assignment.
- Log analyzer that highlights keywords and compresses logs.
- Sorted incident report dashboard for dispatch prioritization.

Grading Breakdown (Total: 50 points)

10 pts	Use of at least 2 graph algorithms (e.g., Dijkstra, Prim, Kruskal)
10 pts	Use of 2 string matching/search algorithms (e.g., KMP, Rabin-Karp)
8 pts	Use of 1 greedy algorithm and 1 dynamic programming algorithm
6 pts	Use of at least 1 sorting algorithm (e.g., Quick Sort, Merge Sort)
6 pts	Presentation: clarity, teamwork, real-world impact
5 pts	Demonstration of working application with I/O
5 pts	Code quality: comments, organization, file naming

Bonus: +2 pts for integrating compression or visualizing graph algorithms with real maps

Final Notes and Tips

- Make sure each student is contributing a core module: algorithms, integration, or UI.
- Reuse and modularize your past code submissions. Cite the file names used.
- Present real data (even simulated) and walk through input/output during your PPT.
- You are encouraged to use visuals, flowcharts, screenshots, or demos.
- Each group will receive peer feedback from classmates on clarity and originality.

Presentation Day: Week 14 in class
Submission: Upload PPT and source files via Canvas by 11:59PM on presentation day.

Daily Project Updates via Discord

To ensure steady progress and team collaboration, each group leader must:

- Post a daily update on our class Discord channel `#project-progress`.
- Include the following in each post:
  - What tasks were completed today?
  - Who worked on what?
  - What is planned for tomorrow?
  - Any blockers or questions?
- Use the following format:

```
Day X — Group Y
Work Completed:
  – Alice implemented Dijkstra from Dijkstra_Test.py
  – Bob worked on KMP string matcher using KMP_test.py

Next Steps:
  – Charlie will integrate QuickSort from QuickSort_02212025.py

Issues:
  – Need help visualizing Huffman from Huffman_Code_03082025.py
```

- This is mandatory and graded indirectly under collaboration and presentation sections.
- Missed daily updates will affect the final presentation score.

Consistent updates will not only reflect professionalism but also make your final demonstration stronger and more organized.

Submission Checklist and Social Media Engagement

Final Deliverables (Due on Presentation Day)

Each group must submit the following items via Canvas:

1. **PowerPoint Presentation (PPTX or PDF)**:
   - Max 10 slides.
   - Must explain the motivation, algorithm integration, sample input/output, challenges, and team contributions.
2. **Source Code (.zip file)**:

- Include all your '.py' files and relevant test data.
- Include a README.txt file listing team members and how to run the project.

Social Media Engagement (Required for Each Student)

To simulate real-world portfolio and networking practice, each student is required to create a post on **LinkedIn** highlighting the project. This will:

- Help build your digital footprint.
- Showcase your algorithm engineering experience.
- Share your collaboration and coding strengths.

Post Guidelines:

- Make your post public on LinkedIn.
- Include a photo/screenshot or your team working/demoing the code.
- Mention the algorithms used (e.g., Dijkstra, KMP, Huffman Coding, etc.)
- Use at least **3 hashtags**, such as:
  - #CPSC335
  - #AlgorithmEngineering
  - #GraphAlgorithms
  - #PythonProjects
  - #ComputerScience

Submission: Include screenshot of your LinkedIn post into the PPT of your Canvas submission.

The goal is to promote your work and learn how to present it to peers, recruiters, and the tech community. Treat this as a mini digital portfolio project.