CSC 403 Data Structures II
Term Project
Professor Eric J. Fredericks
Fall 2024

Project Overview

Each student will design and implement a project in Java that applies at least two data structures, including at least one data structure from DS2. The project should demonstrate how these data structures can solve a real-world problem or create a practical tool.

This is an individual project worth 20% of your final grade. You may collaborate and discuss issues pertaining to your project, get help, etc. with others in the class or even outside the class. However, the work you submit must be your own original work. If you have any questions about what's allowed while collaborating with others, please speak with me.

Students can choose any project idea as long as it incorporates one data structure from DS2 (trees, especially search trees/balanced search trees, hash tables, k-d trees, graphs, ternary search tries/R-way tries) and an additional data structure from either DS1 (arrays, linked lists, stacks, queues, heaps, union-find) or DS2. Data structures not covered in either course may also be used to satisfy the second requirement. If you would like to use something from outside DS1 or DS2, or something we mention in either course without going into depth during lecture, please make sure to discuss this with me early in the project.

The focus of this assignment is to use data structures in a programming project. You should use the data structures provided with the course code archive at the beginning of the quarter. If you would like to use a data structure from outside the course, you may implement it yourself (with appropriate unit tests), or you may use an external library.

If you use an external library (any code besides your own or the code provided for this course or for DS1), you must get approval from me.

You do **not** need to build a flashy graphical user interface (GUI). You are welcome to build one if you prefer, but you may also use the command line and/or file input if desired. The code in the **stdlib** package in your course code archive has classes called **StdIn** and **StdOut**, which you can use to create simple menus and text-based interaction.

If you would like access to materials for topics in the course that pertain to your project before we cover them, please reach out to me. I'll be happy to share them with you.

Each student should implement a different project. If more than one student wants to do the exact same project, I will reach out to you to discuss alternatives.

Several project ideas are included at the end of this handout.

Learning Goals

Key Concepts:

- Apply data structures learned in DS2 to practical problems.
- Integrate an additional data structure not covered in this course.
- Analyze and demonstrate the efficiency of the chosen data structures.

Skills Development:

- Problem-solving with data structures
- Integration of different data structures
- Application of asymptotic analysis in real-world contexts
- Proficiency in Java programming

Project Requirements

Project Choice: Students are free to choose any project idea that meets the following criteria:

1. **Data Structures**: The project must use a total of two data structures, with at least one from DS 2.
2. **Functionality**: The project should demonstrate meaningful use of the selected data structures.
3. **Documentation**: Students must provide clear documentation and analysis of their project.
4. **Programming Language**: The project must be implemented in Java.

Project Phases and Milestones

See the section on Assessment below for details.

Phase 1: Proposal
- *Timeline: September 10-16*
- ***Deadline** for submission on D2L: September 16*

Phase 2: Design
- *Timeline: September 17-30*
- ***Deadline** for submission on D2L: September 30*

Phase 3: Implementation
- *Timeline: October 1 – October 28; Suggested Completion: October 28*
- ***Do not wait until the last minute. You should work on this throughout the month of October.***

Phase 4: Testing and Integration
- *Timeline: October 29 – November 4; Suggested Completion: November 4*

Phase 5: Documentation and Presentation
- *Timeline: November 5-11*
- ***Deadline** for Zoom meeting (if this option is chosen): November 11*
- ***Live Presentations** (if this option is chosen) will be delivered in class on November 12*

Assessment Criteria

Proposal (5%)

- **Written Proposal:** The proposal must clearly outline the project idea, the chosen data structures, and a high-level plan for their integration and use.

I will provide feedback on your project proposal by Tuesday, September 24 and let you know if your project proposal is approved. If I do not approve your proposal, we will need to discuss it and agree on appropriate changes.

Design Document (10%)

- **Description of the application you will develop.** What problem will your application solve?
- **Justification of Choice of Data Structures (5%)**: The design document must clearly explain the rationale for selecting each data structure, including an analysis of their time and space complexities in relation to the problem being solved.
- **Interaction and Integration Plan (5%)**: The document should provide an explanation of how the application will interact with the chosen data structures, including any algorithms that will be utilized. This section should demonstrate an understanding of how the data structures complement each other in solving the problems addressed by the application.

Functionality and Data Structure Usage (35%):

- **Correctness and Efficiency (20%)**: Does the project correctly utilize the chosen data structures, and does the application function as intended? Are there any redundant or inefficient uses of data structures?
- **Appropriate Use of Data Structures (15%)**: Does the project demonstrate an accurate understanding of the data structures' strengths and limitations? Does it apply them in a way that showcases their benefits for the specific problem?

Integration and Interaction of Data Structures (10%):

- **Integration:** Does the project meaningfully integrate components utilizing the data structures in a way that enhances the solution? Is there a clear and logical flow in how data is managed, processed, and retrieved across the different structures?

Code Quality and Clarity (10%):

- **Readability and Organization (5%)**: Is the code well-organized, readable, and modular? Are variables and functions appropriately named to reflect their purpose?
- **Informative Comments (5%)**: Are comments provided where necessary to explain non-trivial sections of code, particularly where data structures and algorithms are applied? Comments should focus on explaining the reasoning behind complex implementations rather than stating the obvious.

Testing and Validation (10%):

- **Comprehensive Testing (10%)**: Are the chosen data structures and their uses thoroughly tested with a variety of test cases, including edge cases? Are there automated unit tests to verify correctness?

Documentation (10%):

- **Instructions for Developers (5%):** Are there appropriate instructions provided so that another developer looking at the project codebase for the first time can get it running, get automated tests working, and use the application within 15-20 minutes?
- **User Documentation (5%):** Is there appropriate user documentation indicating how to use the application? This does not have to be extensive, but it should provide enough information to help users effectively use the application for the problem it solves.

Presentation (10%):

Prepare a 10-minute (maximum) presentation to showcase the project, demonstrate its use, and provide a tour of the codebase. Students may choose to present to the class live in the classroom or on Zoom or meet with the me privately to share their presentation.

- **Project Presentation (5%)**: Is the project presented clearly, with a focus on the problem, chosen data structures, and their application? The presentation should demonstrate a clear understanding of the concepts showcased by the project.
- **Defense of Choices (5%)**: During the presentation, can the student effectively defend their choice of data structures? This includes answering questions about performance, potential alternatives, limitations, and optimizations.

Guidelines and Support

Programming Environment and Source Code

- **I will give you instructions for how to get started with your programming environment.** We'll cover that during Week 3.
- **You are responsible for backing up your work.** If you fail to make backups and you lose your work, you will not be able to submit any code, and risk failing the assignment.
- **You may wish to use GitHub or Bitbucket for cloud-based source code management.** This will alleviate the need for you to make separate backups. I will help with this by providing some basic instructions (more later). If you use a cloud-based solution, your repository should be private and you should allow me to have read access. It's really easy for me to look at code and provide feedback to you iteratively if you decide to use this approach.
- If you do not wish to use GitHub or Bitbucket, you can submit Zip files with *compiling* Java source code, including automated tests, and documents (please submit PDF files, and avoid Word documents – sometimes my Linux machine/LibreOffice Writer can't see information in Word docs). *Please don't submit .class or .jar files in your zips.*

Support and Feedback

- **Office Hours**: See me during office hours if you need help at any point during the quarter. I am happy to talk through challenges you are facing, look at your code and provide feedback, help with tests that you can't get working, etc. I can also arrange to meet with you outside of office hours if necessary.

- **Discussion Forums**: I will create a channel in our course Discord server for students to talk about project issues, get help from me or other students, etc.

Example Project Ideas

Here are several project ideas. You are not limited to these ideas – you're welcome to come up with your own.

Game Development:

1. **Maze Generator and Solver**:
    - DS 2 Data Structures: Implement pathfinding algorithms using graphs or heaps.
    - Additional Data Structure: Use arrays or linked lists to represent the maze grid.
2. **Inventory Management System for an RPG Game**:
    - DS 2 Data Structures: Use hash tables for quick access to item properties.
    - Additional Data Structure: Use stacks for managing item usage and undo actions.
3. **Sudoku Solver**:
    - DS 2 Data Structures: Use backtracking with recursion, leveraging trees or stacks.
    - Additional Data Structure: Use arrays or hash tables for managing constraints.

Computer Networking:

1. **Network Packet Analyzer**:
    - DS 2 Data Structures: Use hash tables or trees for organizing packet data.
    - Additional Data Structure: Use queues to manage packet flow.
2. **Network Topology Visualization Tool**:
    - DS 2 Data Structures: Use graphs to represent network topology.
    - Additional Data Structure: Use linked lists to store network nodes and connections.
3. **Bandwidth Monitoring System**:
    - DS 2 Data Structures: Use heaps to manage and analyze bandwidth usage.
    - Additional Data Structure: Use queues to process network traffic data.

Software Engineering:

1. **Task Scheduler**:
    - DS2 Data Structures: Use a hash table to store tasks for quick lookup and management by their unique IDs.
    - Additional Data Structure: Use a priority queue (min-heap) to manage and schedule tasks based on their priority efficiently.
2. **Version Control System**:
    - DS 2 Data Structures: Use hash tables for quick retrieval of file states.
    - Additional Data Structure: Use stacks to manage commit history.
3. **Dependency Manager**:
    - DS 2 Data Structures: Use graphs to handle project dependencies.
    - Additional Data Structure: Use queues to process build tasks.

Bioinformatics:

1. **DNA Sequence Alignment Tool**:
    - DS 2 Data Structures: Use tries or hash tables for sequence comparison.
    - Additional Data Structure: Use arrays or linked lists for sequence manipulation.
2. **Protein Structure Prediction**:
    - DS 2 Data Structures: Use graphs to model protein folding patterns.
    - Additional Data Structure: Use stacks or queues to manage sequence data.
3. **Genomic Variant Finder**:
    - DS 2 Data Structures: Use hash tables for variant storage and lookup.
    - Additional Data Structure: Use arrays to store and compare genomic sequences.

Cybersecurity:

1. **Intrusion Detection System**:
    - DS 2 Data Structures: Use tries or red-black trees for anomaly detection.
    - Additional Data Structure: Use hash tables to store suspicious patterns.
2. **Password Strength Analyzer**:
    - DS 2 Data Structures: Use hash tables for storing password patterns.
    - Additional Data Structure: Use arrays to evaluate password policies.
3. **Phishing Email Detector**:
    - DS 2 Data Structures: Use tries for pattern matching in email content.
    - Additional Data Structure: Use hash tables to store known phishing signatures.

Data Management and Search:

1. **Autocomplete System**:
    - DS 2 Data Structures: Use tries for autocomplete functionality.
    - Additional Data Structure: Use hash tables or balanced trees for managing search terms.
2. **Text Search Engine**:
    - DS 2 Data Structures: Use tries to index and search text.
    - Additional Data Structure: Use hash tables for document metadata.
3. **Data Deduplication Tool**:
    - DS 2 Data Structures: Use hash tables to identify and remove duplicate data entries.
    - Additional Data Structure: Use arrays or linked lists to store and manage unique records.
4. **Geospatial Data Management and Search System:**
    - DS 2 Data Structures: Use K-d Trees to manage and query multi-dimensional geographic data points.
    - Additional Data Structure: Use hash tables to store and retrieve metadata associated with each geographical point, such as names, types (e.g., restaurant, park), and/or ratings.

Graph Theory and Network Analysis:

1. **Social Network Analysis**:
    - DS 2 Data Structures: Use graphs to model social connections.
    - Additional Data Structure: Use union-find to detect and manage connected components.
2. **Minimum Spanning Tree**:
    - DS 2 Data Structures: Use graphs and heaps to find the minimum spanning tree for some real-world application.
    - Additional Data Structure: Use union-find to manage and merge sets during the MST algorithm.
3. **Shortest Path Finder**:
    - DS 2 Data Structures: Use graphs and priority queues (heaps) for shortest path algorithms for some real-world application.
    - Additional Data Structure: Use arrays or linked lists to store path distances and predecessors.