# Assignment 4
# Heaps and Compression

CSIS 3475

# Assignment

- Download **Assignment 4.zip** and import it into an Eclipse workspace using the standard instructions.
- Finish the MinHeap project
- Finish the Huffman Coding project
- Submit the completed projects using the standard submission instructions
    - Export the projects to a zip archive named **Assignment 4 YourName.zip** where YourName must be your first initial and last name.
    - You MUST use the submission instructions exactly or you will lose marks.
    - You MUST name the archive correctly or you will lose marks.
    - For example, for Michael Hrybyk
        - Assignment 4 MHrybyk.zip

# Project 1 – MinHeap Implementation

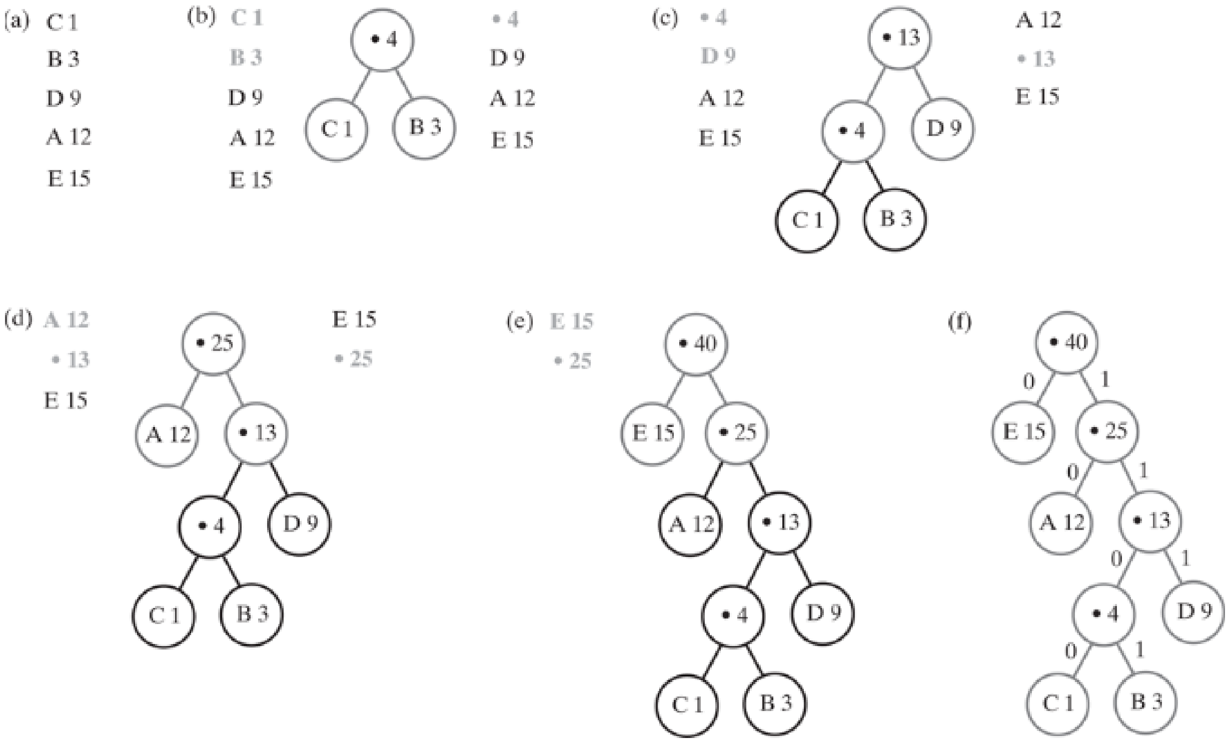
- Complete the MinHeap class, which implements the MinHeapInterface.
    - MinHeap creates a heap with the smallest element as the root.
    - Complete all methods
    - You will need to code a reheap() method
    - This is exactly equivalent to the MaxHeap class covered in the textbook and in class. The only difference is the comparisons are reversed.
    - Test thoroughly with the driver provided.
- HeapPriorityQueue class is provided with a test driver.
    - This uses MinHeap
    - This will be used as a priority queue for the next project, Huffman Coding.
    - Test driver is provided for this class as well.

# Project 2 – Huffman Coding

- Complete Carrano Chapter 25 Project 10, pages 706-707, with some differences
    - You only need to display modified Huffman codes, which in this case always have a 1 as the first code (basically, the root).
    - No need to actually compress a file
    - Create output that matches that for the test input files, dicionary.txt and FrequencyCounterData.txt
- Complete all of the code in HuffmanCodingDemo
    - The main program, including file i/o set is already done.
    - The following methods need to be completed
        - readFile() – reads all characters from a text file and places them in a character map
        - displayCharacterMap() – displays the map, which consists of a character and another object containing data about the character (frequency, Huffman code, …)
        - toPQ() – converts the character map to a priority queue. The character with the lowest frequency has the highest priority. You must use HeapPriorityQueue for this (which in turn uses MinHeap)
        - toTree() – converts the priority queue to a Huffman tree (see the textbook for a picture and explanation).
        - setCodes() – traverses the tree, and sets the bits for a Huffman code in each node
    - All methods have extensive Javadoc comments, please study these.

# Project 2 – About Huffman Codes

- Huffman codes are created from a binary tree
  - leaves are letters with lowest frequencies.
  - Root nodes of subtrees have values of the sum of the frequencies of leaves.
- Codes are created by denoting a 0 for the left child and a 1 for the right child, then preorder traverse the tree from the top to create the code



The steps in creating a binary tree for Huffman coding

# Project 2 – Huffman Coding – CharNode Class

- This class has already been created.
- It is similar to BinaryNode covered in the textbook and class
- It has fields for a character, its frequency, and its Huffman code.
- Also contains pointers to right and left child.
- Basic setter and getter methods are provided.
- You do not need to change this class in any way. You must use it in the HuffmanCodingDemo program.

# Project 2 – Huffman Coding - Testing

- Two input files are provided
  - dictionary.txt (from the Anagrams assignment)
  - FrequencyCounterData.txt (from the sample program covered in the Dictionaries class).
- A correct program's output will match dictionaryOutput.txt and FrequencyCounterDataOutput.txt exactly.
- Feel free to test against other text files

# Project 2 – Huffman Coding - Notes

- **Every** character in a text file is counted.
  - Includes newlines, tabs, spaces, …
- If you can't get MinHeap working completely, you may use the Java Library to implement a priority queue.
  - If you do it this way, you must create a priority queue class that implements PriorityQueueInterface.
  - You will lose some marks if you do it this way, however.
- The project in the textbook can serve as help in understanding Huffman codes using a binary tree.
  - See also https://en.wikipedia.org/wiki/Huffman_coding
  - Our Huffman codes are preceded by a 1 (which can be stripped off if we were to write it to a file)
  - Follow the project description, but you should not actually write code to compress and decompress the file (although this would be awesome if you can figure it out).

# Grading

| Item | Marks |
|---|---|
| Project properly named and submitted | 0.1 |
| All code properly formatted and commented | 0.1 |
| MinHeap Implementation | 2.3 |
| Huffman Coding Implementation | 2.5 |
| | |
| Total | 5.0 |