CS 3305/W02: Data Structures

Assignment 6 –Heaps-Trees Check D2L for Due Date (100 points total)

GENERAL SUBMISSION REQUIREMENTS

Upload all files individually as specified, not as zip files, to Assignments in D2L. Do not email files.

It's best if your program compiles, runs and produces the correct output.

Ensure you have the correct file name(s), and author header, as specified in the Assignment.

Always use meaningful labels for prompts, inputs, and outputs.

Always use comments, indentation and whitespace as shown in examples.

Note: Never hard-code test data in the test program, unless explicitly stated in the assignment. Always allow the user to enter the test data using a menu option.

Objectives The purpose of this lab is to reinforce Heaps and Tree concepts in Java

Assignment 06 – PART 1 Heaps (50 points)

Part 2 - Objective of Part 1 is to reinforce understanding of Heap data structures.

A max-heap is a heap in which each node is greater than or equal to any of its children. A min-heap is a heap in which each node is less than or equal to any of its children. Min-heaps are often used to implement priority queues (see Bonus assignment at end of document). Revise the Heap class in Listing 23.9 (which starts on p.878) which implements a **MAX**-heap and convert it to a **MIN**-heap.

Use the following logic:

1. Write a main method that will accept 5 numbers, and put them in a min-heap
2. Remove them from the min heap, printing one at a time as they are removed, to show that the list is sorted

Hint: the textbook example is for a MAX heap; you must alter that example to reflect a MIN heap – think about it !

No files or data are provided for this part of the assignment.

Do not forget to include author header in each submitted file as shown, no header, no points!

// Name: <your name>
// Class: CS 3305/ put your section number after the /
// Term: Semester Year
// Instructor: Sharon Perry
// Assignment: 6-Part-1-Heaps

DELIVERABLE INSTRUCTIONS – PART 1

Capture a **READABLE** screenshot(s) of your program output and paste into a word/pdf document. Readable means readable! Screenshots ***should not be an entire desktop*** – use some type of snipping tool. After your output screenshots, copy and paste the source code for your program into the word/pdf doc. Save doc as a file named LastName-A6-Part-1-Heaps.docx or .pdf. Last step is to upload everything to D2L, to clarify upload

1. word/pdf doc that contains output screenshot and copy/pasted source code; AND
2. your .java file that contains your source code.

Only your latest submission is kept so if you update a file and want to resubmit you must upload all files. Submit everything to the assignment submission folder in D2L by the due date posted in D2L.

No zip file or email submissions are accepted.

MAKE SURE YOUR CODE HAS COMMENTS ! We are getting submissions without comments in the code. No comments in source code = (-30) points *per each Part of the assignment.* This penalty grows as time goes on !!

Late penalties of 10 % per day are in effect for this assignment.

Assignment 06 – PART 2 TREES (50 points)

Part 2 - Objective of Part 2 is to reinforce understanding of binary tree data structures.

You are going to write a program that uses a binary tree to translate morse code into characters. You may use the Java Libraries to solve this problem. We recommend using java.util.*

Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes, or dits and dahs, respectively. Morse Code refers to a system for representing letters of the alphabet, numerals, and punctuation marks by an arrangement of dots, dashes, and spaces, known as the Morse Code Alphabet. A picture of the alphabet is shown below. In telecommunication a space is designated by "time units" between characters so for our purposes, we will designate a space with NULL.

Program Binary Tree

Create a binary tree to establish a morse-code table, and implement encode and decode methods.

Build the binary tree, by reading from the file morse.txt.

You will encode and decode messages by searching on THAT tree. Look at the morse-code alphabet image below with yellow background, and the image of the binary tree below that (that is what your binary tree should look like when you build it).

Notice how if you search LEFT on the tree it registers a dot.

Notice how h translated to morse code moves left 4 times on the tree, therefore h is represented as 4 dots **....**

Data

For your data you need letters and a Boolean (NULL), indicating that the node is empty, since not all nodes encode data. Here a sequence of dots " . " and dashes " - " are used to represent morse-code. Use spaces to separate morse-code 'letters'.

Read the morse-code data from the attached file called morse.txt . This file contains letters (lowercase), a space, and the morse code. *While our images all show capital letters for morse code and the binary tree, for simplicity you may assume that all letters are lower case only.*

Class and Methods

You need to define a class, MorsecodeTree. This class should contain the following methods:

String encode method - takes a string of characters and encodes it as morse-code. Letters in either upper or lowercase get translated to morse, and the dot " . " and dash " - " pass through unchanged. Any other character is not passed to the output string. Use the space to separate individual morse code letters. **Hint**: you may want to do one that encodes one character. To accomplish all of this you need to search the tree.

String decode method - takes a string of space-separated morse-code letters and produces a string with the corresponding alphabetical letters. The produced string has all lowercase letters spaces between words**.** You can safely assume the input string contains only dot " . " and dash " - " and spaces. If the combinations of dot " . " and dash " - " for a given character is not valid, no corresponding character should be put on the output string. **Hint**: To accomplish all of this you need to search the tree.

Program Input / Output

Your program should allow the user to select either encode or decode message option. Then, the program will ask for the appropriate message string and print

the appropriate corresponding result. The program should then return to the option to select either encode or decode messages.

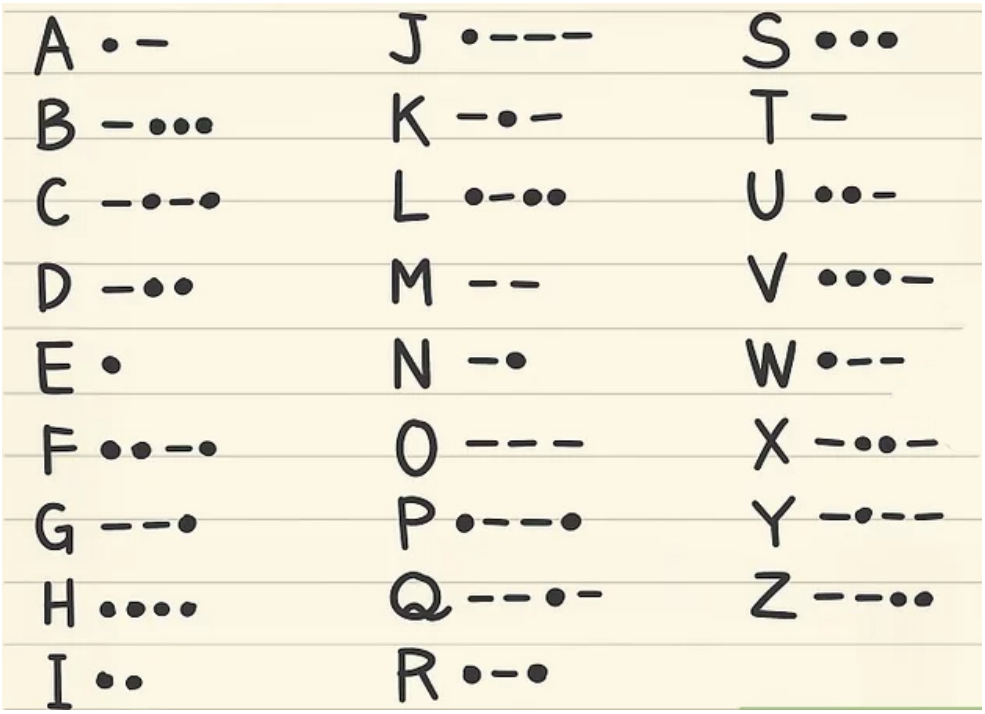
If user selects option to encode a message, user enters an alphabetic string and system prints the morse-code message result.

If user selects option to decode a message, user enters a morse-code message and system prints the result translated into an alphabetic message.

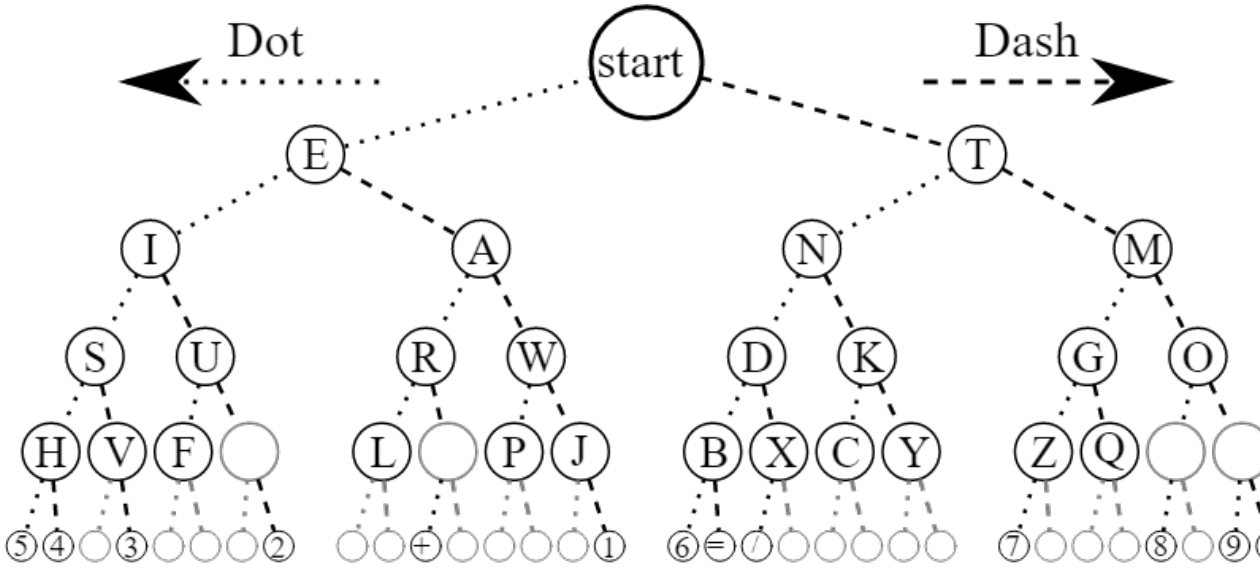
Getting Started

1. Review the Morse code alphabet and binary tree images below.
2. Review the attached file called morse.txt which represents the letters of the alphabet that you will use to build your binary tree. Build your tree using the morse.txt file (note: you can use an array or linked list). Morse.txt is essentially a depth first search reading of the binary tree structure - and nulls represent end nodes - ie if you get to 2 nulls the node above it is a leaf node - for example h is a leaf node
3. Your program should read this data from a file to build the tree.
4. Once you build your binary tree, it has a form shown in the binary tree image on the next page.
5. Allow users to select a message option (encode or decode) and have your program print the translated message out to the screen.

   MORSE CODE ALPHABET - these images are capital letters but it doesn't matter – assume capital and lower case are the same



BINARY TREE FOR MORSE CODE



Do not forget to include author header in each submitted file as shown, no header, no points!

// Name: <your name>
// Class: CS 3305/ put your section number after the /
// Term: Semester Year
// Instructor: Sharon Perry
// Assignment: 06-Part-2-Trees

DELIVERABLE INSTRUCTIONS – Part 2

Capture a **READABLE** screenshot(s) of your program output and paste into a word/pdf document. Readable means readable! Screenshots ***should not be an entire desktop*** – use some type of snipping tool. After your output screenshots, copy and paste the source code for your program into the word/pdf doc. Save doc as a file named LastName-A6-Part-2-Trees.docx or .pdf. Last step is to upload everything to D2L, to clarify upload

1. word/pdf doc that contains output screenshot and copy/pasted source code; AND
2. your .java file that contains your source code.

Only your latest submission is kept so if you update a file and want to resubmit you must upload all files. Submit everything to the assignment submission folder in D2L by the due date posted in D2L.

No zip file or email submissions are accepted.

MAKE SURE YOUR CODE HAS COMMENTS ! We are getting submissions without comments in the code. No comments in source code = (-30) points *per each Part of the assignment.* This penalty grows as time goes on !!

Late penalties of 10 % per day are in effect for this assignment.

Assignment 07 – PART 3 HEAPS * * * * * BONUS * * * * * OPTIONAL * * * * (10 points)

For the Bonus in Assignment 6, redo the problem in Assignment 4, Part 2, about Priority Queues but implement your solution with a HEAP. A heap is a special type of binary tree where, every parent is less than both children (aka a min-heap) or greater than both children (aka a max-heap). Heaps are considered an efficient data structure to use for priority queues.

Assignment 4, Part 2 is reprinted below in dark green italics for your review.

Write a program to simulate checkout lines at a grocery store. There will be multiple queues, one for each check out line. For this exercise, you may assume there are 5 check out lines. You can use an array of queues to simulate the checkout lines.

Use the following logic

1. *Generate 5 customers and enqueue them, one customer in each queue*
2. *Each new customer ready to check out chooses the shortest line*

Customers enter the check out queues randomly, and then each time a customer is generated that customer chooses the shortest line.

If the lines are equal, then the first available line is chosen. Each transaction takes a random amount of time to complete. Print each action taken with Queue number, to the display.

For your output show the queues with customers and activity, showing changes in each queue.

You can capture all of the output at the end of the program running. Program should list actions that have been performed and Queue numbers.

Be sure to limit your program to ensure that it does not run forever.

Do not forget to include author header in each submitted file as shown, no header, no points!

```
// Name: <your name>
// Class: CS 3305/ put your section number after the /
// Term: Semester Year
// Instructor: Sharon Perry
// Assignment: 06-Part-3-Extra-Credit
```

DELIVERABLE INSTRUCTIONS – Part 3 - Bonus Assignment 6 – 10 pts (Optional)

Capture a **READABLE** screenshot(s) of your program output and paste into a word/pdf document. Readable means readable! Screenshots ***should not be an entire desktop*** – use some type of snipping tool. After your output screenshots, copy and paste the source code for your program into the word/pdf doc. Save doc as a file named LastName-A6-Part3-Bonus.docx or .pdf. Last step is to upload everything to D2L, to clarify upload

1. word/pdf doc that contains output screenshot and copy/pasted source code; AND
2. your .java file(s) that contain your source code.

Only your latest submission is kept so if you update a file and want to resubmit you must upload all files. Submit everything to the assignment submission folder in D2L by the due date posted in D2L.

No zip file or email submissions are accepted.

MAKE SURE YOUR CODE HAS COMMENTS ! We are getting submissions without comments in the code. No comments in source code = (-10) points – *in other words no credit for the bonus.*

Late penalties of 10 % per day are in effect for this assignment.