CS3460: Data Structures Fall 2021
Assignment 09: Fun with Graphs
Due Date: Monday, November 22; 23:00 hrs
Total Points: 25

In this assignment, we will gain practice with graph traversal and shortest path algorithms. In particular, we will solve water jugs with a depth-first search traversal, and the word ladder problem using breadth-first search. Both traversals are performed on "implicit graphs" – graphs whose nodes and edges are revealed while traversing it.

1. **Filling Water Jugs (10 points):** You are standing next to a river with a head of cabbage and two water jugs, which have integer sizes $A \leq 1000$ and $B \leq 1000$. In order to boil the cabbage for your dinner, you would like to measure out exactly C units of water. Please write a program that, given A, B and C, computes how to do this, or determines that the task is impossible. For example:

```
Enter A: 3
Enter B: 4
Enter C: 5
Yay! Found a solution.
Fill Jug 1 [a = 3, b = 0]
Pour Jug 1 -> Jug 2 [a = 0, b = 3]
Fill Jug 1 [a = 3, b = 3]
Pour Jug 1 -> Jug 2 [a = 2, b = 4]
Empty Jug 2 [a = 2, b = 0]
Pour Jug 1 -> Jug 2 [a = 0, b = 2]
Fill Jug 1 [a = 3, b = 2]

Enter A: 3
Enter B: 6
Enter C: 5
Impossible!
```

To solve this problem, you should use a depth-first search through a graph where each node corresponds to a pair of integers (a, b), indicating that you are in the state where jug 1 contains a units of water and jug 2 contains b units of water. You want to start from the state $(0, 0)$ where both jugs are empty, and your goal is to reach a state (a, b) with $a + b = C$. There are 3 possible actions you can take to move between states: filling one of the jugs to its capacity, emptying out one of the jugs, or pouring the contents of one jug into another (until the first becomes empty or the second reaches its capacity). If your program finds a solution, it should print out a step-by-step transcript of the solution such as the one above.

Please name your program `Jugs.java`. You can use a double dimensional array to represent the states you have visited. Note that the graph is implicit, meaning the graph is discovered as we run depth-first search. You also need to use "backpointers" to help print the solution. Every time a new state is visited, say (x, y), the previous state (px, py) is recorded. This information will be used to print the solution. A string can also be attached to print meaningful information about the action that lead to state (x, y).

2. **Word Ladder (15 points):** Consider a word ladder that can be formed between two 4-letter words.

$$\textbf{lose} \rightarrow \textbf{lost} \rightarrow \textbf{lest} \rightarrow \textbf{best} \rightarrow \textbf{beat}$$

Each word is formed from the previous word by changing exactly one letter. In this problem, we would like to find the shortest possible set of words that transforms the start word into the end word. Note that all valid words should belong to the dictionary (file `dictionary4`) and valid letters are small case a through z.

The solution is to perform a breadth-first search through a graph where each node corresponds to a word, and edges correspond to possible transitions from a given word to the next. To solve this problem effectively, we will need two data structures. The first is a `StringMap`, that is a hash map of a set of words. Recall that a hash map is a data structure that can store a collection of key-value pairs. The hash map will be used for two things – to store valid words from `dictionary4`, and secondly, to keep track of visited and previous nodes during breadth-first search. We can use the `key` field to store the visited words, and the corresponding `value` field to store "backpointers" – the word that is visited previously during breadth-first search. This is used to print the path (or the sequence of words) from the start to the end word. The files `StringNode.java` and `StringMap.java` are already fully-functional. Please go through these files to see all the supported functions, and their specifications.

We also need a simple FIFO queue data structure. The file `Queue.java` contains startup code to implement a queue using circular arrays. Note that each element in `Queue` is of type `QNode`, which contains two fields – `dist` and `word`. The `dist` field stores the shortest distance of the current `word` from the start word while performing breadth-first search. The queue is initialized with an array of 1000 elements, and therefore needs code in `enqueue` to expand the array whenever more room is needed. Please complete all the required functions and thoroughly test your implementation.

We can then start implementing the word ladder in `WordLadder.java`. This file currently loads all the 4-letter words from the dictionary onto a string map. It also acquires user input for the start and end words. Please complete the code so that it prints the shortest sequence of words to go from the start to the end words, or prints such a sequence is impossible. Two sample inputs and output are provided below.

```
Enter the start word: lose
Enter the end word: beat

Yay! A word ladder is possible.
lose
lost
lest
best
beat

Enter the start word: drug
Enter the end word: high
Duh! Impossible.
```

Submission: Please submit all the files as a single zip archive `h09.zip` through ASULearn. The zip archive should only contain the individual files of the assignment, and these files should not be inside folders. Moreover, please do not include other extraneous files. Only include all files that belong to your solution.

Input/ Output Instructions: For all programs, until and otherwise stated, we will be taking the input from standard input (`System.in`) and will be sending the output to standard output (`System.out`). Since we will be using an automatic grading system, please make sure that your output format exactly matches the description above. So make sure that there are no extraneous output in terms of spaces, newlines and other characters.

Notes on Coding: Please do not include user-defined packages in your code. Your code should run in the Unix/Linux machine using the commands `javac` and `java`.

Please note that you are **not allowed to use Java Collections** which contain pre-defined libraries for many of the data structures that we learn in class. The purpose of these assignments is to build these data structures from first principles. Programs that includes these objects will receive 0 points.