Version: 4-23-2018 9:45 PM

CS 2223 D18 Term. Homework 5

Homework Instructions

- This homework is to be completed individually. If you have any questions as to what constitutes improper behavior, review the examples as I have posted online http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/d18/#policies.
- Due Date for this assignment is 9PM April 30th. Homeworks received after 9PM receive a 25% late penalty. Homeworks received after Midnight will receive zero credit.
- Submit your assignments electronically using the canvas site for CS2223. Login to canvas.wpi.edu and locate HW5. You must submit a single ZIP file that contains all of your code as well as the written answers to the assignment.
- All of your Java classes must be defined in a packager USERID where USERID is your CCC user id.
- Submission information is found at the end of this document.

Primary Instructions

Submit your written answers in a file writtenAnswers.txt that you submit with your assignment.

Q1. Breadth-First Search on undirected graph **[35 pts]**

Q2. Counting Sort example **[30 pts]**

Q3. Graph Properties **[35 pts]**

Bonus Question: Word Pyramid **[1 pt]**

Version: 4-23-2018 9:45 PM

Q1. Word Ladders: Breadth First Search Exercise (35 pts)

A word ladder is a word game invented by Lewis Carroll. A word ladder puzzle begins with two words of the same length. To solve the puzzle, you must find a sequence of other words to link the two, in which two adjacent words in successive steps differ by one letter.

For example, given the words COLD and WARM, the following is a sample word ladder:

COLD → CORD → CARD → WARD → WARM

Note how each successive word differs by exactly one letter from the prior word. While the sequence of words can be quite long, the goal of this question is to come up with the shortest path given a dictionary of valid English words (such as found in **`words.english.txt`** which you can find in the repository).

Here are the assumptions you can make:

1. All words in the word ladder are just four letters long.
2. You are to use a table `SeparateChainingHashST<String, Integer>` *`table`* to store the mapping from a four-letter word to an integer, representing that word's vertex in the undirected graph.
3. You are to use a table `SeparateChainingHashST<String, Integer>` *`reverse`* to store the reverse mapping from the integer vertex id to the four-letter word.

Modify the existing **`WordLadder`** class to solve this problem.

Version: 4-23-2018 9:45 PM

Q2. Counting Sort (30 pts)

We have claimed that no comparison-based sorting algorithm can perform in time better than Θ (n log n). There are other approaches that, for specific domains, can perform better.

You are given an array of size *N* that contains integer values between 0 and *k*, for some fixed *k* value that you know in advance. Your task is to sort these values. It is possible to do this in time Θ (n + k). Consider the following input with n=14 and k=3:

0	1	2	1	2	1	0	3	0	2	1	2	1	3

You can count there are three 0, five 1, four 2, and two 3 values. With this information in hand, you can immediately create the sorted array by copying these values back into the array as follows:

0	0	0	1	1	1	1	1	2	2	2	2	3	3

That is, from left to right, the first three cells are 0, the next five are 1, the next four are 2, and the last two are 3.

(a) [**15 pts**]. Complete the provided **`CountingSort`** class to implement counting sort

(b) [**10 pts**]. Explain why your code achieves Θ(n+k) performance.

(c) [**5 pts**]. Conduct timing experiment to sort elements and compare against MergeSort for N=4096 to N=524,288.

Version: 4-23-2018 9:45 PM

Q3. Graph Properties (35 pts)

We have begun introducing proofs into the lectures, and the textbook also contains samples throughout the different chapters. It is time for you to try your hand at a proof as it relates to an undirected graph.

(a) **[15 pts]** Prove that every connected graph of N vertices contains some *safe vertex*, **v**, whose removal (including all edges incident **v**) will not disconnect the graph.

You may make use of the following facts:

- A graph with no vertices is also, by definition, considered to be connected
- A graph with a single vertex is, by definition, connected. Note that there are no edges in such a graph

Hint: Depth First Search should be useful. Consider a vertex whose adjacent vertices are all marked.

(b) **[10 pts]** Modify the method **`findSafeVertex()`** in the **`Graph`** class that returns a vertex in a connected graph which can be deleted and enable the graph to remain connected. Note if the graph is initially not connected, then -1 must be returned. You should check this first.

```
public int findSafeVertex () { … }
```

(c) **[10 pts]** The *eccentricity* of a vertex, *v*, is the length of the shortest path from that vertex to the furthest vertex from *v* in the graph. Naturally the graph must be connected for this property to make sense. With this definition in place, we can define the *diameter* of a graph to be the maximum eccentric of any of its vertices. Modify this method in the **`Graph`** class, which returns -1 if the graph is not connected.

```
public int diameter () { … }
```

Version: 4-23-2018 9:45 PM

BONUS question: Word Pyramid [1 pt]

A word pyramid is a word game. A word pyramid puzzle begins with a word of N letters. To solve the puzzle, you must find a sequence of other words, of decreasing size in length, until you reach a word with just a single letter. To produce the next word in the sequence, remove a letter and form a new word from the remaining letters

For example, given the starting word, RELAPSE, the following is a sample word pyramid:

RELAPSE → PEARLS → SPEAR → PEAS → SEA → AS → A

Note how each successive word differs by exactly one letter from the prior word (and is an anagram of the remaining letters). Each of the subsequent words must be a valid word (such as found in **`words.english.txt`** which you can find in the repository).

Here are the assumptions you can make:

1. The length of the initial word will be seven characters or less.
2. You are to use a table `SeparateChainingHashST<String, Integer>` *`table`* to store the mapping from a word to an integer, representing that word's vertex in the undirected graph.
3. You are to use a table `SeparateChainingHashST<String, Integer>` *`reverse`* to store the reverse mapping from the integer vertex id to its associated word.

This is a bonus question, so you are on your own. Feel free to create a new class based on the Q1 Word Ladder class. Here is my sample output from a run:

```
Enter word to start from (all in lower case):
relapse
relapse
serape
spear
spar
spa
pa
a
```