Version ~~4-22-2019: 8:45 AM~~ 4-27-2019 2:35 PM

CS 2223 D19 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/d19/#policies.
- Due Date for this assignment is 2PM April 29th. Homeworks received after 2PM receive a 25% late penalty. Homeworks received after 6PM 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 **[50 pts]**

Q2. Status Injective **[25 pts]**

Q3. Graph Properties **[25 pts]**

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

Bonus Question: Spreadsheet **[1 pt]**

Version ~~4-22-2019: 8:45 AM~~ 4-27-2019 2:35 PM

Q1. Word Ladders: Breadth First Search Exercise (50 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. Use an AVL<String> tree to store all these words.
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<Integer,String>` *`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. Here are some hints:

1. Use an AVL<String> tree (as implemented in `algs.days.day18.`AVL) to store all four-letter English words. You can load up the "words.english.txt" file and add all four-letter words to this AVL tree. Because it is an AVL tree, it will self-balance as words are inserted in ascending order.
    a. As each word is added to the AVL tree, add an entry into *`table`* and *`reverse`*.
2. Next, create a graph, G, with the same number of vertices as there are words in *`table`*.
3. Now for the final trick, add an undirected edge (u, v) to this graph G if two words (W_u and W_v) differ by just a single letter.

A final hint: how to process all pairs of words in the AVL tree? Consider the following:

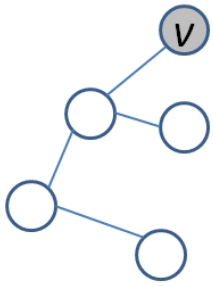
```
String min = avl.min();
for (String w1: avl.keys()) {
  for (String w2: avl.keys(min, w1)) {   // will do one more check than necessary...
    …
  }
}
```

This will allow you to process all pairs of words (W_u and W_v). Note that you need to write logic to determine if two Strings differ by just a single letter.

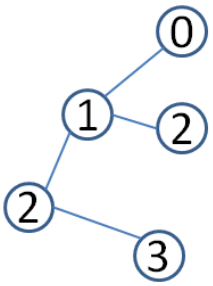
Version ~~4-22-2019: 8:45 AM~~ 4-27-2019 2:35 PM

Q2. Graph Properties (25 pts)

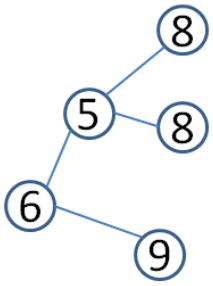
You are given a simple, undirected connected graph G = (V, E). For any vertex, v, in this graph you can compute its *status*. The *status* of *v* is the sum of the shortest distances to every vertex in the graph (recall that the distance of a vertex to itself is zero). Consider the graph below and designated vertex *v*.



Status[v] = 0+1+2+2+3 =8



Shortest distances from *v*



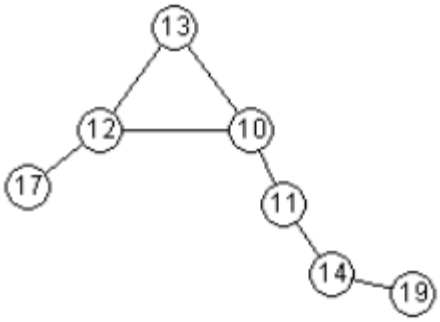
All Status Values

The status of *v* is 8, since that is the accumulated sum of the shortest distance from *v* to every other vertex in the graph. Note that this is **not** a status injective graph because two vertices have a computed status of 8 as shown above.

Q2.1. [15 pts] Modify the `Graph` class to complete the implementation of the `status(v)` method which returns the computed status for a given vertex, *v*, in the graph.

```
public int status(int v) { … }
```

Q2.2. [10 pts] A graph is classified as being *status injective* if the calculated status values for all nodes in the graph are different integers. To convince you that such graphs exist, consider the following graph which draws each vertex with its computed status value. As you can see, all of these values are distinct, thus this is an example of a status injective graph.




```
public boolean statusInjective() { … }
```

Version ~~4-22-2019: 8:45 AM~~ 4-27-2019 2:35 PM

Q3. Graph Properties (25 pts)

Q3.1 [10 pts] The **diameter** of an undirected **graph** is the maximum number of edges between any pair of vertices. To find the **diameter** of a **graph**, first find the shortest path between each pair of vertices. The greatest length of any of these paths is the **diameter** of the **graph**.

```
public int diameter () { … }
```

You will need to modify the `Graph` class to complete this implementation. Note that in doing so, you should also complete the following method that determines whether an undirected graph is connected.

```
public boolean connected() { … }
```

Q3.2 Copy `algs.hw5.SearchCompare` into your `USERID.hw5` package and complete it. This class takes in an undirected graph, `G`, and a source vertex, `s`. We have covered in class how **BREADTH FIRST SEARCH** can compute a `bfsDistTo[v]` that records the shortest path from `s` to each vertex, `v`, in terms of the total number of edges used in the path from `s` to `v`. Well, you can also compute a `dfsDistTo[v]` that computes a similar value for a **DEPTH FIRST SEARCH**. This search is blind and makes no guarantee regarding the length of the path it chooses from `s` to each vertex, `v`.

Define `excess[v] = dfsDistTo[v] – bfsDistTo[v]`. This value is always non-negative (can you see why?) You must modify `SearchCompare` so it computes the sum of `excess[v]` for all vertices in a graph from a designated vertex, `s`.

```
public static int excess (Graph G, int s) { … }
```

Once completed, you can execute SearchCompare on the sample `tinyG.txt` file (which you can copy into your MyCS2223 project from the Algos-D19 project. I referred to this graph on lecture on day 20). This graph has 13 vertices, and it should compute the following for a start vertex of 0:

```
bfsDistTo:   [0, 1, 1, 2, 2, 1, 1, +∞, +∞, +∞, +∞, +∞, +∞]
dfsDistTo:   [0, 1, 1, 4, 2, 3, 1, +∞, +∞, +∞, +∞, +∞, +∞]
```

As you can see, each `dfsDistTo[v]` value is greater than or equal to each `bfsDistTo[v]`, and the sum total of these differences is four, and this is the value that should be returned by calling `excess(G, s)`.

Q3.2.1 [5 pts] Produce correct answer for Excess on tinyG.txt file.

Q3.2.2 [10 pts] For N in {4, 8, 16, 32, …, 1024} create a complete graph of N vertices – that is, every vertex is connected by an undirected edge to every other vertex in the graph. As I have mentioned in class, the total number of edges on a complete graph of N vertices is N*(N-1)/2. For the complete graph of N vertices, G, your program should compute `excess(G, 0)` and output a table that looks like the following:

N	Excess
4	3
8	21
16	105
32	465
64	…
128	…
256	…
512	…
1024	…

Q3.3 [1 pts] Bonus Question. Can you come up with a general formula, f(N) that computes the computed `excess(G, 0)` for a complete graph?

Version ~~4-22-2019: 8:45 AM~~ 4-27-2019 2:35 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:

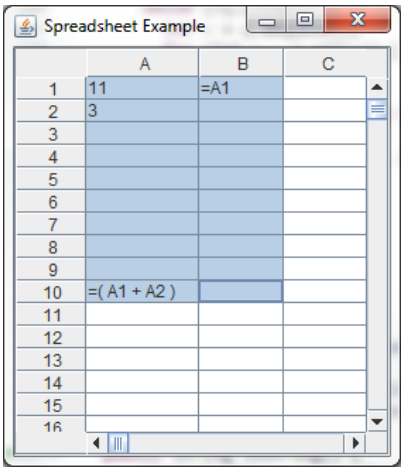
```
Enter word to start from (all in lower case):
relapse
relapse
serape
spear
spar
spa
pa
a
```

Version ~~4-22-2019: 8:45 AM~~ 4-27-2019 2:35 PM

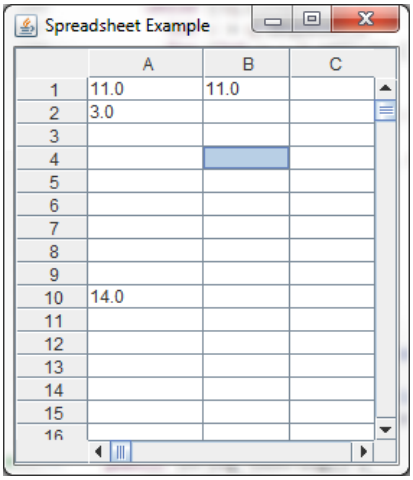
BONUS question: SpreadSheet Application [1 pt]

Complete the sample spreadsheet application provided in the `algs.hw5.ss` package. You should copy this entire package into your USERID.hw5 package and only modify the Spreadsheet class file.

When it is working, you can enter in values such as the following. Note that A1 contains the value 11 and cell B1 is a copy of that value. Note that cell A1 uses the infix expression evaluator (modified for this assignment) to compute value of arbitrary computations with cell references and numbers.



And when not-selected, you will see the computed values.



Note: this bonus question is not for the faint-hearted and is only worth a single point.