Version: 3-May-2021 11PM

CS 2223 D21 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/d21/#policies.
- Due Date for this assignment is **6PM Tuesday May 11th** which is the day before the end of the term.
- 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

Q1 WordZipper (50 pts)
Q2 Shortest Distances (20 pts)
Q2.1 Bonus (1 pt)
Q2.2 Bonus (1 pt)
Q3 Popular Symbol Table (30 pts)
Bonus Word Zipper (1 pt)

From **`algs.hw5.submission`** package, copy all classes into your USERID.hw5:

- **`PopularSymbolTable, Q2, TestPopular, WordZipper`**

Version: 3-May-2021 11PM

Q1. Word Zippers: Breadth First Search Exercise (50 pts)

Look at the following arrangement of words:

cold → old → wold[1] → wod[2] → woad[3] → wad → ward → war → warm

A *Word Zipper* puzzle begins and ends with two four-letter words (cold and warm in this example). To solve the puzzle, you must find a sequence of different words to link the two, alternating between four letter and three letter words (no word can appear twice in the same word zipper):

- To transform a four-letter word into a three-letter word, you can only drop one letter
- To transform a three-letter word into a four-letter word, you can only add one letter

In all cases, the left-to-right ordering of the unaffected letters remains unchanged.

Note how each successive word differs by exactly one letter from the prior word (either adding or removing). 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 zipper are just either three or four letters long. Use an AVL<String> tree to store these 3- and 4-letter words.
2. Use a `SeparateChainingHashST<String,Integer>` *`map`* to store the mapping from a word to an integer, representing that word's vertex in the undirected graph.
3. Use a `SeparateChainingHashST<Integer,String>` *`reverse`* to store the reverse mapping from the integer vertex id to the word.

Copy `algs.hw5.submission.WordZipper` into your `USERID.hw5` package and modify it to solve this problem. Here are some hints:

1. Use an `AVL<String> tree` (as implemented in `algs.days.day18.AVL`) to store all three- and four-letter English words. Use the **`Dictionary`** class to retrieve the words in the dictionary.
    a. As each word is added to the AVL tree, add an entry into *`map`* and *`reverse`*.
2. Next, create a graph, G, with the same number of vertices as there are words in *`map`*.
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, according to the rules above. Complete the implementation of the **`addOne()`** and **`removeOne()`** methods and use them appropriately in your solution.

Hint: There should be a total of 7,224 three- and four-letter words in the dictionary, which means your graph should have this many vertices. Once you have computed the edges that must exist by the definition of the game, there will be 9,496 edges. Note that you should try to be careful not to add

---

[1] in Britain, (often in place names) a piece of high, open, uncultivated land or moor.
[2] Obsolete variation of WOAD.
[3] An annual Old World plant in the mustard family, formerly cultivated for its leaves that yield a blue dye.

(unnecessary) multiple edges for the same pair of vertices. HOWEVER, if you do, it won't change the results of the computations so in the end, it won't have a major impact on the assignment.

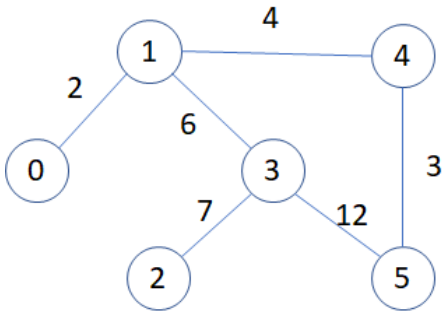
Q1.1 Bonus Question (1pt)

What pair of four-letter words (w1 and w2) demands the longest Word Zipper given the words in the dictionary? This will be the pair of words for which the SHORTEST PATH between w1 and w2 is longer (or equal) to any other pair of words. What is the actual Word Zipper for these words (in either direction is fine)?

Version: 3-May-2021 11PM

Q2. Shortest Distance (20 pts)

Given the Massachusetts highway map data, find two vertices in the graph such that **the shortest distance between them** is **greater than any other pair of vertices in the graph**.



In other words, can you find two locations in Massachusetts such that, using the available map data, you've computed the shortest total distance in terms of accumulated mileage, and **no other pair of vertices demands a longer trip**.

In the above graph, once you have completed the shortest distance between any two vertices, you will discover that the **longest of these shortest distances** is 19, which is the path from 2 to 5.

The output of your program on Massachusetts Highway data should look like this:

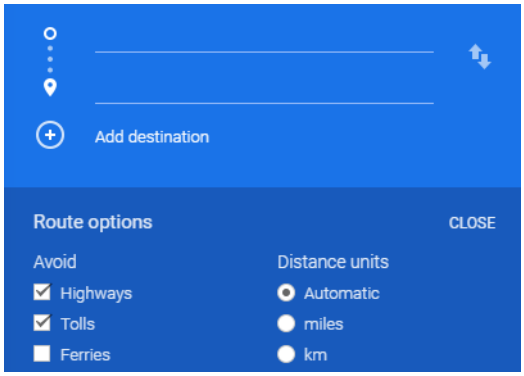
```
vertex1 is XXXXX (LABEL) @ GPS
vertex2 is YYYYYY (LABEL) @ GPS
Total mileage: ZZZZZ
```

Note that ZZZZZ is the total accumulated mileage in miles. A correct solution will output a value between 200 and 300 miles.

To solve this, you must use the Floyd-Warshall algorithm. The **`FloydWarshallSolutionAnimation`** class can be used to visualize the resulting path and you might use it to confirm your results.

Q2.1 Bonus Question (1pt)

When you enter in the GPS coordinates for these two vertices into https://google.com/maps for directions, are any of the proposed paths shorter than the one you found? If theirs is shorter, why didn't your algorithm find it? If yours is shorter, why didn't Google report the one you found?



Q2.2 Bonus Question (1pt)

In the https://google.com/maps interface, you can request to avoid highways and tools. Why does Google's search return a shorter distance than the one you found? What is your explanation?

Version: 3-May-2021 11PM

Q3. Data Type with Performance Guarantees (30 pts)

You have seen several data types that provide performance guarantees when implemented with the proper data structures. For this question, you are to design and implement a `PopularSymbolTable` data type, which is like a symbol table that maintains a set of (key, value) pairs, but there is added functionality. For simplicity, each key is just an **`Integer`** type and each value is also an **`Integer`**. Aside from this limitation, the key (or value) can be any 32-bit integer, including negative values.

Copy `algs.hw5.submission.PopularSymbolTable` into your `USERID.hw5` and complete it.

As you should expect, operations **`put(key, value)`** or **`get(key)`** or **`remove(key)`** exist. There is also a **`size()`** method to return the number of keys in the Symbol Table.

The following two enhancements do not change the functionality, but make it easier to test:

- Make sure that **`put(key, value)`** returns **`true`** if the key is newly being added; otherwise it returns **`false`** if the key already exists and the new **`value`** is now associated with it.
- Make sure that **`remove(key)`** returns **`true`** if the key was actually removed; if the key does not exist in the symbol table, then return **`false`**.

The most important change, however is the **`Queue<Integer> reverseMatch(value)`** method, which returns in ascending order those **`keys`** that map to the exact same **`value`**. For example, consider:

```
PopularSymbolTable pst = new PopularSymbolTable();
for (int i = 0; i < 10; i++) {
  pst.put(i, i % 3);
}
```

The above makes the following associations, for example **`pst.get(7)`** is 1.

0 → 0	1 → 1	2 → 2	3 → 0	4 → 1
5 → 2	6 → 0	7 → 1	8 → 2	9 → 0

With this `PopularSymbolTable`, **`pst.reverseMatch(2)`** returns those keys that map to the value of 2: above this would be the values 2, 5, and 8, and the returned **`Queue`** must contain values in ascending order (so the first value to be dequeued would be 2).

For full credit:

- **`put`**, **`get`** and **`remove`** must perform in O(log N) in the worst case.
- **`size`** must perform in O(1).
- **`reverseMatch(value)`** must perform in O(K + log N) where K is the number of keys that match to the given value. Note K <= N in all cases. If, in fact, all keys in the symbol table map to the exact same value, then K = N and the performance is O(N).

Copy `algs.hw5.submission.TestPopular` into your `USERID.hw5` and execute it. It should pass all functional tests BUT ALSO provide output that you should save in a `writtenQuestions.txt` file to include as evidence you have hit your performance target.

Version: 3-May-2021 11PM

The output of TestPopular should look like below. Note that SearchS should only have `0.000` since the time to complete is O(log N) which is simply below the visible three decimals of precision.

```
In the following table:
-----------------------
BuildT is the build time, which should be O(log N)
SearchS should be essentially 0 since it does reverseMatch when K is small
SearchL should be O(N) since it does reverseMatch when K is N/2
N       BuildT  SearchS SearchL
2048    0.047   0.000   0.031
4096    0.094   0.000   0.016
8192    0.188   0.000   0.063
16384   0.344   0.000   0.141
32768   0.750   0.000   0.234
65536   1.672   0.000   0.563
131072  3.797   0.000   0.984
262144  9.047   0.000   2.594

In the following table:
-----------------------
BuildT is the build time, which should be O(log N)
SearchT should be O(N) since reverseMatch returns K=N values
N       BuildT  SearchT
2048    0.031   0.031
4096    0.063   0.063
8192    0.141   0.109
16384   0.328   0.234
32768   0.688   0.484
65536   1.594   1.031
```

Version: 3-May-2021 11PM

BONUS question: Extended Word Zipper [1 pt]

Take the basic Word Zipper puzzle and, this time, add all words of up to seven letters to the mix. However, now, an edge exists if you can add (or remove) a VALID word from amongst the original letters to form a new word. If you do this, there will be 80,490 vertices in your graph and a total of 180,556 edges (at least, that's what I calculated…).

To make this interesting, elimate any attempt to just add/remove prefix of "UN" or "RE". Otherwise, you could have GOOD → UNGOOD → UN → UNEVIL. If you put this change into effect, then the total number of edges drops from 180,556 to 178,343 edges.

RESTAFF → AFF[4] → TARIFF → RIFF → SHERIFF

From "RESTAFF" remove "REST", leaving "AFF". Now add the word "TRI" to create "TARIFF" (note how the letters interleave). Now remove the word "TA" to produce "RIFF" and then add "SHE" to make SHERIFF.

TRIAL → RA → TERRANE[5] → ERR → ERROR

From "TRIAL" remove "TIL", leaving "RA". Add the word "TERNE" to create "TERRANE". Remove the word "TANE" to leave "ERR". Then add "OR" to create "ERROR".

Here's another one, without the "UN-" prefix capability

ABLE -> USABLE -> ULE[6] -> UNABLE

This is a bonus question, so you are on your own. Feel free to discard everything you did for the WordZipper and do what you can to beat the time **117.875** seconds for computing "RESTAFF to SHERIFF" when launching the executable from scratch. Can you beat the record?

LET'S GO!!!
SpeedRunning 2.0 is ON

[4] From the Scottish: means "of".
[5] A fault-bounded area or region with a distinctive stratigraphy, structure, and geological history.
[6] From OED, "an obsolete variant of OIL or OWL."