UNIVERSITY OF VIRGINIA, DEPARTMENT OF COMPUTER SCIENCE

Hash Tables - Implementation and Word Puzzle Search

Dr. Mark R. Floryan

August 20, 2019

1 SUMMARY

For this homework, you will be building a custom hash table class, and using it to perform a word search. The overview of this homework is as follows:

1. Download the provided starter code.
2. Implement the HashTable.java class.
3. Test the correctness of your hash table using the provided tester.
4. Implement the word search solver by using your hash table implementation.
5. Optimize your hash table to run as quickly as possible on the large word search grids provided
6. **FILES TO DOWNLOAD:** HashTables.zip
7. **FILES TO SUBMIT:** HashTables.zip

1.1 HashTable.java

First, you will implement a hash table class. This class has two generic arguments K (the key) and V (the value). To handle these, we will have our table internally store HashNode objects. This code is provided to you and a summary is shown below:

```
public class HashNode<K,V> {
    private K key; private V value;

    //Constructor
    public HashNode(K key, V value);

    //HashNodes are "equal" if both keys are equal
    public boolean equals(Object other) {
        return this.key.equals(((HashNode<K,V>)other).key);
    }

    public K getKey() {return this.key;}
    public V getValue() {return this.value;}
    public void setValue(V newValue) { this.value = newValue; }
}
```

Notice that two HashNode objects are considered to be "equal" to each other if the key in each is equal. Thus, you can use the provided HashNode.equals() method to examine equality between HashNode objects directly.

Using this object, you will implement the HashTable.java class which implements the Map interface shown below. You may use *any collision resolution strategy* you would like when implementing this class.

```
public interface Map<K,V> {

    public void insert(K key, V value);
    public V retrieve(K key);
    public boolean contains(K key);
    public void remove(K key);
}
```

Once you have implemented your HashTable class, there is a provided tester that will test some of the operations of the table to ensure they work. **This tester IS NOT a thorough test, but rather a sanity check that the table seems to be working**. You should write your own tests as well to ensure your table is working correctly.

1.2 WORD SEARCH SOLVER

Next, you will be writing some code within the WordPuzzleSolver.java class. This class reads in a two-dimensional array of characters, and looks for valid English words within the puzzle. (see Wikipedia for more details). You will be provided with two input files, a **dictionary text file** and a **grid text file**. The dictionary file enumerates a list of valid English words. Your goal is thus to find each of these words in the word puzzle. The grid text file specifies the number of rows and cols in the grid and then enumerates the characters of the grid all on a single line.

Some other notes / requirements about this part of the homework:

1. You should instantiate your hash table and insert each of the words from the dictionary. In order to this, you will need to read input from a file. See the course java cheat sheet for an example of how to read from a file.

2. You need to check words that start at ANY starting location in the grid, moving in any of the eight directions (North, NorthWest, etc.), and check any words of length $3 \leq n \leq 22$. Make sure you do not move off the end of the grid when looking up words.

3. Your output should match the format shown in the provided starter project under *input/3x3.out.txt*. The order the words are printed is not important, but would be nice to make grading easier.

4. the *input/* folder provides several grids and dictionaries to test your code with. Your code should be correct for all combinations and also be reasonably fast (less than 10 seconds on the largest grids). How fast can you make it?