Homework 1

GWU CSCI 1112 - Fall 2023

September 8, 2023

1 Introduction

This homework is intended to reinforce your understanding of working with one-dimensional arrays and to encourage you to think algorithmically. Your task is to develop an algorithm that solves the problem described using the provided framework.

1.1 Deadline

September 16, 2023 at 11:59pm

1.2 Submission

You must create a `.zip` file containing all of the files necessary to compile and run this project. This includes all `.java` source files that you develop or use and any assets, *e.g.* `words`, that contain data. You must submit a `.zip` file containing all of these files to blackboard by the deadline.

Your `.zip` file must be named using the following naming convention: `<yourNetId>-hw-01.zip`
For example, my NetId is `jrt`. If I submitted the homework, I would name my `.zip` file: `jrt-hw-01.zip`.

1.3 Grading Rubric

- 30% For successful compilation of unit tests AND Scramble game
- 20% For execution without exception of both unit tests and Scramble
- 30% For passing all unit tests (10% for each block of unit tests)
- 10% For sufficient comments
- 10% For consistent coding style
- 10% (Bonus) For optimally efficient implementation

1.4 Comments

I have provided relevant comments to document the file (a file header), the function signatures (function block comments), and inline comments. These comments are a model that you should follow in the future when developing files or functions on your own. This level of comments will be expected in future assignments and future grading will assess all three of these criteria as a part of commenting.

You must substitute your name in the file header and provide any inline comments to document code that you contribute to your solution. If you add your own functions, you must include a function block comment following the model illustrated by the predefined function declarations.

1.5 Plagarism

We will use a set of automated tools specifically designed to analyze code for plagiarism. If you copy code from another source, classmate or website, there is a very high probability that these tools will flag your work as plagiarized. You are permitted to discuss the problems at a high level; however, you must code your own solution. If you do not share code or outright borrow code from a website, you will have no problem with the plagiarism filter. If your work is found to be plagarized, your grade will be reduced to a zero and it will be treated as a violation of academic integrity.

1.6 Use of Generative AI

As the goal of the homework is to practice Java programming and to encourage the development of soft skills such as analysis and debugging, Generative AI is prohibited. A correct solution without understanding is a useless exercise. Per the syllabus, you will be audited on your solution using questions of quizzes and interviews. If you are incapable of defending your solution in these audits, your grade for this homework will be reduced to a zero. If multiple homework grades are affected this way, it will be treated as a violation of academic integrity.

1.7 Use of Imported Java Types

A goal of this class is to develop an understanding of how to build data structures. We will be developing our own classes for ArrayLists, LinkedLists, Stacks, Queues, BinaryTrees, etc. For this reason, you are not permitted to import any of the existing classes from any library unless explicitly stated.

2 Scramble

"Scramble" attempts to model well-known, tile-based spelling games like "Scrabble" or "Words with Friends". In a tile-based spelling game, each tile represents a character from the alphabet and each tile can only be used once per word. The basic goal is for a player to "draw" a "hand" of tiles from a "bag" and then spell the highest value word possible from their hand. This process may be repeated for a number of rounds according to a variety of rules or constraints specific to a particular game's rules. The winner is typically determined to be the player with the highest cumulative score from all rounds.

Once a tile is used, that tile cannot be reused again for the same word. This does not mean a letter can only appear once in a word, but it does mean that for a player to use a character more than once, there must be at least the number of tiles with that character available in the player's hand. So, for example, assuming a player has a hand containing "MAEZ", that player can play "MAZE" but cannot play "AMAZE" because that player's hand only contains one 'A'.

Scoring for a tile-based, spelling game may be as simple as counting the number of tiles used to spell the word or may have more complex rules like summing up a point total derived from point values associated with specific characters. In Scrabble, each character has an associated point value illustrated on the tile, so computing scores under Scrabble rules involves totalling the point values of all tiles in the word by associated each character with its point value. Here are the standard point values for Scrabble tiles:

A	1	B	3	C	3	D	2	E	1
F	4	G	2	H	4	I	1	J	8
K	5	L	1	M	3	N	1	O	1
P	3	Q	10	R	1	S	1	T	1
U	1	V	4	W	4	X	8	Y	4
Z	10								

Based on this table of point values, the score for playing "MAZE" would be:

$$\begin{array}{cccccc} \text{M} & \text{A} & \text{Z} & \text{E} & & \\ 3 & + 1 & + 10 & + 1 & = & 15 \end{array}$$

There are also natural limits imposed by the tiles. A tile may only be played once, there are a limited number of tiles in the bag (98 character tile and two blank tiles which we do not model in our game), and each character appears on a fixed number of tiles, so there are a limited number of times a character can be used until there are no more tiles with that character on them available. Here are the number of each character's tiles in the set of tiles.

A	9	B	2	C	2	D	4	E	12
F	2	G	3	H	2	I	9	J	1
K	1	L	4	M	2	N	6	O	8
P	2	Q	1	R	6	S	4	T	6
U	4	V	2	W	2	X	1	Y	2
Z	1								

Since there is only one 'Z' in the bag, there would be no way to spell the word "PIZZA" as that would require the player to draw two 'Z' tiles but this is impossible since there is only one 'Z' in the bag. Conversely, it is possible to spell "AMAZE' because there are at least two 'A' tiles, at least one 'M' tile, at least one 'Z' tile, and at least one 'E' tile in the bag.

2.1 Framework

There are a number of classes, assets, and methods provided to support this program and you will only need to write three specific methods for your solution. This section will detail all of the existing elements of the framework that are there to support your program and their roles as well as the specific methods that you are to implement. A detailed explanation of the methods you need to implement will be provided in the next section of this document.

The files distributed in this framework are `Scramble.java`, `TileGame.java`, `UniformRandom.java`, `UnitTests.java`, `Utilities.java`, `words`, and `WordTool.java`. You must submit your versions of ALL of these files for a complete solution. You should not submit additional files, specifically, do not submit any `.class` files.

You will not need to interact with the `words` or `WordTool.java` files. `words` is an asset, *i.e.* a data file, containing a list of valid English words and is used to create a dictionary in the program, and it is critical to the overall execution to the program since this is an English dictionary based game. `WordTool.java` is a class designed to load a dictionary from the hard drive into memory as an array of String data and the methods in this file are already used at the necessary points in the framework.

You will also not need to interact with the `UniformRandom.java` file. This file comes directly from the course website and it allows a programmer to generate random numbers from a uniform distribution. This random number generator is used when drawing a hand which is code that is already provided to you.

To keep things simple, all character data will be exclusively limited to legal uppercase alphas, *i.e.* 'A' to 'Z', so we need a way to clean our data to purge any non-alpha characters from strings or to format any lowercase alphas as uppercase characters. In the `Utilities.java` file, you will find a useful method called `clean` that will allow you to ensure a String contains no illegal characters for this program. `clean` takes a reference to a String as input, removes any non alpha characters from the input string, formats any of the remaining alpha characters in the string as uppercase characters, and returns a reference to this newly cleaned string. You may need to use this method whenever you extract a word from the dictionary.

You will primarily work with the `TileGame.java` file. The `TileGame` class contains the methods that encapsulate logic for a tile based, English spelling game. This is the class where you will make your additons to the program and this class will be detailed in the next section.

You will also interact with the `UnitTests.java` and `Scramble.java` files. `UnitTests.java` contains a set of validation tests that are designed to test functions in the program in a variety of ways to help validate the integrity of the methods. The `Scramble.java` file contains a demo program that shows how the game is played assuming you have successfully implemented the unfulfilled methods in `TileGame.java`. Both of these programs are executable and are compiled against and use the rest of the Java files in their programs.

2.2 The `TileGame` class

The `TileGame` class is the heart and soul of the game logic that makes this tile-based spelling game work. There are a number of attributes and methods in this class to support the game. You will need to implement three methods in this class to complete the game. This section helps detail all the existing attributes and methods to give you an understanding of how the class supports the game.

`static final int[] points`

The `points` array represents the model set of points for each character tile. This array is used whenever the score for a tile and a word are computed. The methods that use this array are already provided. You will not need to use this array yourself; however, you might want to investigate how it is used if you want to understand how the game is fully put together.

Note that this is a zero based ordinal array where 0 maps to the character 'A' and 25 maps to the character 'Z'.

`static final int[] tilebag`

The `tilebag` array represents the model count of tiles available for each character. Note that the first cell of the `tilebag` array contains a 9. This corresponds to the number of 'A' tiles available in a game of Scrabble. When one 'A' is drawn, this decreases the number of 'A' tiles available by one.

Note that this is also a zero based ordinal array where 0 maps to the character 'A' and 25 maps to the character 'Z'.

Since this is model data, it represents the default, initial "state" of the game. We may need to return to this inital state if the game is restarted, so we need to protect this data; however, we also need to use the data. To this end, we will make a deep copy of the `tilebag` array whenever tiles are drawn. If you examine the `drawHand` method, you will see how the copy of the `tilebag` data is used.

`public static final int MAXTILES = 98;`

`MAXTILES` is the count of the number of tiles in the game.

`public static int getTilePoints( char ch )`

`getTilePoints` takes a character `ch` as input and cross-references that character with the `points` table to determine what the point value for a given alpha character is and returns the associated point value. This assumes that the input `ch` is an uppercase alpha only and returns a non-zero value in that case. If `ch` is not a valid uppercase alpha character, `getTilePoints` returns zero.

`public static int getWordScore( char[] word )`

`getWordScore` takes a reference to a character array `word` as input, computes the score for the entire word based on the individual tile points, and returns the total score. It is dependent on `getTilePoints` to compute the points associated with each character in the array and it is subject to the same constraints, all characters in `word` are restricted to uppercase alphas.

`public static char[] drawHand( int count )`

`drawHand` takes an integer `count` as input representing the maximum number of tiles to draw, generates a hand consisting of uppercase alphas constrained by the limits according to the maximum instances of a tile as defined by the data in `tilebag`, and returns a reference to the newly drawn hand.

3 Requirements

In this section, you will find detailed requirements for the methods that you MUST implement in `TileGame.java`.

`public static int[] copyTileBag()`

This method makes a deep copy of the `tilebag` array and returns the copy of that array. As explained above, the `tilebag` must not be changed as it is the model intial data for when a game is started. We perform a deep copy of the `tilebag` data whenever we draw a hand and then we can manipulate the copy with no fear that we are corrupting the data used to initialize the game.

Block 1 unit tests found in the `UnitTests.java` file exclusively validate the `copyTileBag` method. If you pass all the tests in Block 1, you can be assured that are performing a deep copy of the `tilebag` using this method. You should implement this method first and validate it completely before proceeding on with the remaining methods. If this method is not validated, there will be unexpected issues in the rest of the program.

While debugging, you may need to see more information about the state of the program. You are welcome the add print statements into the UnitTests to peek at the state of various values. However, we will validte your program against the original `UnitTest.java` file, so you will want to avoid changing the logic of the tests as that may render different results from the validation that will be performed on your submission.

You are NOT allowed to use the java utility library to perform the deep copy. The goal of this course is for you to develop a fundamental understanding of the algorithms and operations discussed. It is not the goal for you to use other libraries at all. Using the `java.util` library to copy an array avoids the point of the course and prevents you from developing the level of understanding expected. One of the specific goals for this exercise is to develop a complete understanding of deep copy. In general, assume that you cannot use a method from an external file unless it is explicitly stated that it is allowed. The exception in this homework is that you are allowed to use the `toCharArray` method from the String library to extract a character array from an existing String object since all of the methods operate on character arrays.

`public static boolean canSpell( char[] hand, char[] word )`

`canSpell` takes a reference to a character array called `hand` and a reference to a character array called `word` as input. It determines whether the characters in `hand` can be used to spell the word specified in `word`. Keep in mind that each letter in `hand` can only be used once in the attempt to spell the `word` according to rules listed above. So if `hand` contains "EZMA", the `word` "MAZE" can be spelled, but the `word` "AMAZE" cannot. Returns `true` if the `word` can be spelled by the characters in `hand`; otherwise, returns `false`.

There are some edge cases that must be addressed and these edge cases are clearly enumerated by the unit tests. In general, you need to handle what happens if any of the parameters are `null`. I suggest you keep these tests simple.

Block 2 unit tests found in the `UnitTests.java` file exclusively validate the `canSpell` method. These tests are comprehensive and should validate your `canSpell` method. Since `canSpell` is used by the `getBestWord` method below, you should develop and validate `canSpell` completely before attempting to implement `getBestWord`.

`public static String getBestWord( char[] hand, String[] dictionary )`

`getBestWord` takes a reference to a character array called `hand` and a reference to an array of strings called `dictionary` as input. It searches the dictionary for the word with the highest point value as determined by `getWordScore` that can be spelled by the characters in `hand` and it returns a reference to that word. If no word can be spelled by the characters in `hand`, it returns a `null` reference.

This process requires you to iterate over all words in the dictionary which means you will need to loop through all words in the dictionary. For each word, you will need to test it using `canSpell` and you may need to compute its

score using `getWordScore`.

Note that the methods in the class expect references to `char` arrays as input, yet the values stored in the dictionary have String types. You can convert a String from the dictionary to a character array using this example:

```
char[] word = dictionary[i].toCharArray();
```

Again, there are a number of edge cases that must be addressed and these edge cases are enumerated by the unit tests and accompanying documentation. You must handle `null` parameters.

Block 3 unit tests found in the `UnitTests.java` file exclusively validate the `getBestWord` method which has dependencies on `canSpell` and consequently `copyTileBag`.