CMPUT 175 Wi19 - INTRO TO FNDTNS OF CMPUT II

Assignment 4

Assignment 4 - Exercises

Due Date: April 4th, 2019 at 23:55

Percentage overall grade: 5%
Penalties: No late assignments allowed
Maximum Marks: 100

Pedagogical Goal: Experience with Singly Linked Lists, Linear Data Structures, and sorting.

Background:

In this assignment, you will be implementing Word Guess, a variant of the game Hangman. In this game, a word is first randomly chosen. Initially, the letters in the word are displayed represented by "_".

For example, if the random word is "yellow", the game initially displays "_ _ _ _ _ _".Then, each turn, the player guesses a single letter that has yet to be guessed. If the letter is in the secret word, then the corresponding "_" are replaced with the letter. If the letter appears more than once in the secret word, then "_" is replaced with the letter for each instance. For example, if the player guesses the letter "l", then the game's progress will be updated to "_ _ l l _ _". The player has a finite number of guesses to guess all the letters in the randomly chosen word. This finite number of guesses depends upon the word to be guessed. Details will be given later (See Task 3). If the user correctly guesses all the letters in the secret word within the allotted number of guesses, they win. If the player uses up all their guesses before they correctly guess all the letters in the randomly chosen word, they lose.

Tasks to do:

In this assignment, there are four files: *Node.py*, *SecretWord.py*, *WordGuess.py*, and *main.py*. Node.py contains a Node class. SecretWord.py contains the LinkedList class and the SecretWord class. The SecretWord class represents the randomly chosen word that is being guessed in the Word Guess game. It contains an attribute self.linkedList which stores a LinkedList object. The LinkedList class is exactly the same as the Singly-Linked List class presented in the lecture and is composed of Node objects. WordGuess.py includes the WordGuess class, which contains the logic of the Word Guess game. Finally, main.py consists of the functions needed to run this game.

Task 1:

Your task will be to complete the Node class in *Node.py*. The Node class was modelled after the SLinkedListNode class presented in the lecture. The methods setData(element), setNext(reference), getData(), and getNext(), which make up the Node ADT, have already been written for you. Now write the following two additional methods:

- getDisplay(): Returns True if the letter stored in self.data should be displayed when the Word Guess game prints the current game progress. Returns False otherwise.
- setDisplay(newDisplay): Sets whether or not the letter being stored in self.data should be displayed when the Word Guess game prints the current game progress.

Note: You may need to introduce a new class attribute to implement these two methods

Task2:

Now, you will implement the SecretWord class in *SecretWord.py*. The SecretWord class represents the randomly chosen word that is being guessed in the Word Guess game. The SecretWord class contains the attribute self.linkedList, which stores a LinkedList object. The LinkedList class has been included in this file. The LinkedList class is exactly the same as the Singly-Linked List class that was presented in the lecture. All the methods in the List ADT have been included in the LinkedList class and have been written for you. Use the methods in the LinkedList class to help you write the following methods for SecretWord:

- setWord(word): Adds the characters in 'word' to self.linkedList so that the letters maintain their order. Thus, self.linkedList's head should contain the first letter in 'word'.
- sort(): Implement **Insertion Sort** to sort the letters in the current SecretWord object by alphabetical order and return the alphabetically sorted secret word in a new SecretWord object. Your implementation must sort the linked list representation of the secret word -- not the string representation of the secret word. You may find Python's built-in ord() function useful.
- isSolved(): Returns True if all the letters in the secret word have been guessed by the player. Returns False otherwise.
- update(guess): Updates whether or not a letter in self.linkedList should be displayed if it matches 'guess'
- printProgress(): Prints the current game progress in the Word Guess game. For example, if the secret word is "yellow" and only the letter 'l' has been correctly guessed, then this method prints "_

_ l l _ _ ”.

- __str__(): Converts the secret word stored in self.linkedList into a string.

Task 3:

Next, you will implement the WordGuess class in *WordGuess.py*. This class consists of the methods required to play a single Word Guess game. In the version of Word Guess you are creating, the allotted number of guesses is not constant. The allotted number of guesses is a function of the "edit distance” between the randomly chosen word and the alphabetically sorted random chosen word. The "edit distance” between two strings is the minimum number of insertions and deletions required to convert the first string into the second string. For example, the edit distance between "brain” and "crane” is 4:

1. Delete b at position 0
2. Insert c at position 0
3. Delete i at position 3
4. Insert e at position 4

Suppose the randomly chosen word is "brain”. When sorted, "brain” becomes "abinr”. The edit distance between "brain” and "abinr” is 4: abin

1. Insert a at position 0
2. Delete r at position 2
3. Delete a at position 2
4. Insert r at position 4

Once the edit distance is calculated, it is used to calculate the number of allotted guesses. The number of allotted guesses is bounded by the range [5,15] (inclusive). We set the number of allotted guesses to be 2*(edit distance). If this value is less than 5, then the number of allotted guesses is set to 5. If this value is greater than 15, then the number of allotted guesses is set to 15. Using this information, write the following methods:

- __init__(): Set the necessary attributes
- chooseSecretWord(): Randomly choose a secret word from the list of words read in from the input file
- editDistance(s1, s2): Returns the edit distance between s1 and s2, s1 being the string representations of the secret word and s2 the sorted secret word. Remember that the edit distance between s1 and s2 is the minimum number of insertions and deletions needed to convert s1 into s2. You must compute the edit distance using recursion.
- getGuess(): Asks the user to guess a letter they have yet to guess in the secret word. The game keeps querying the player for a letter until they enter a letter that hasn't been guessed yet. A class

attribute will be needed to keep track of which letter have been guessed. The user may also enter the '*' character if they would like a hint that gives information about the secret word. If the user enters '*', they lose one of their allotted guesses.

- play(): Plays out a single full game of Word Guess. Initially, a secret word is chosen at random. Then, the edit distance is computed between the secret word and the sorted secret word. This edit distance is used to compute the number of allotted guesses. Then, while the player hasn't guessed the word and still has guesses available, the game progress is displayed, and the game queries the player for a guess. If the player correctly guesses a letter in the secret word, the number of allotted guesses remains the same. If the player enters a guess not in the secret word, the number of allotted guesses decrements by 1. If the user guesses a letter they have already guessed, they do not lose a guess. The game ends once the player has guessed all the letters in the secret word or has run out of guesses.

Task 4:

Finally, in *main.py*, you must write a main() function that allows multiple games of Word Guess to be played. You must also write a function readWords(filename) which reads in the file 'filename' and stores the secret words as keys and their corresponding hints as values in a dictionary. readWords() must return this dictionary. This dictionary is passed as an argument to the WordGuess constructor. The hint is a single word which helps categorize the secret word that needs to be guessed. For example, in the sample input file below, "apple" is a secret word and its corresponding hint is "fruit" since an apple is a fruit. You should query the player for a file name containing the secret words and hints.

Example of an input file:

```
apple fruit
hockey sport
english language
orange fruit
coldplay band
```

Sample Execution of Word Guess:

```
Please enter a Word Guess input file: hangman_words.txt
A secret word has been randomly chosen!
You have 12 guesses remaining
Word Guess Progress: _ _ _ _ _ _ _
Enter a character that has not been guessed or * for a hint: r
You have 11 guesses remaining
Word Guess Progress: _ _ _ _ _ _ _
Enter a character that has not been guessed or * for a hint: s
You have 11 guesses remaining
Word Guess Progress: _ _ _ _ _ s _
Enter a character that has not been guessed or * for a hint: *
Hint: language
You have 10 guesses remaining
Word Guess Progress: _ _ _ _ _ s _
Enter a character that has not been guessed or * for a hint: e
You have 10 guesses remaining
Word Guess Progress: e _ _ _ _ s _
Enter a character that has not been guessed or * for a hint: n
You have 10 guesses remaining
Word Guess Progress: e n _ _ _ s _
Enter a character that has not been guessed or * for a hint: s
Invalid guess. You have already guessed this letter.
Enter a character that has not been guessed or * for a hint: g
You have 10 guesses remaining
Word Guess Progress: e n g _ _ s _
Enter a character that has not been guessed or * for a hint: l
You have 10 guesses remaining
Word Guess Progress: e n g l _ s _
Enter a character that has not been guessed or * for a hint: i
You have 10 guesses remaining
Word Guess Progress: e n g l i s _
Enter a character that has not been guessed or * for a hint: h
You solved the puzzle!
The secret word was: english
Would you like to play again? (y/n): n
```

Submission Instructions

Please follow these instructions to correctly submit your solution.

- Name your solutions **main.py** ; **Node.py** ; **WordGuess.py** and **SecretWord.py**. You should not change the file names or the names of the methods/functions. You can write helper functions but the methods that have already been commented with TODO should perform the tasks that have been described for them in the instructions.
- Submit your 4 files at the end of this page in a zipped folder using the normal "assignment4_yourStudentID" naming convention.