Lab4 Assignment – Hashing & Correcting spellings

Do these problems in order. The successive problems will require the previous ones to be solved. While implementing/debugging, hard-code the program's input in your Python file. Once your code is working you can prompt the user for input. Needless to mention, do not use Python's in-built dictionary data structure for any part of this assignment!

Problem 1. Hash Table: Implement a Hash table using Chaining to resolve collisions. Assume the keys to be hashed are strings; Implement the `insert, search` and `keys` operations of the hash table. The `keys` function returns a list of all the keys currently stored in the table.

Since Python does not have an in-built array type, you will have to simulate arrays using Lists of a fixed size. (Assume the table size to be 30 for this problem). For e.g. a table (list) of size 30 where all entries are 'None', can be created by the following Python code:

```
T = [None for i in range(30)]
```

Use 'Component-Sum' method to compute the hash code map. For e.g. to hash the string 'cat', find the ASCII(or Unicode) value for the characters 'c','a'&'t' and add them (in this case 99+97+116=312.) You can use the 'ord' function to get the Unicode value of a character: ord('a') returns 97. The compression map of the hash function can simply be the modulo m function, where m is the table size. So if the table size is 30, the string 'cat' will be hashed to the slot 312 % 30 = 12 in the table.

Implement Chaining using Singly Linked Lists. You can resue your code from Lab 2 for this. You might however need to modify class `ListNode` to have **three** attributes: key, value and next (Alternatively, you could keep the ListNode class unchanged, by taking the Listnode value attribute to be a (key,value) tuple.)

Problem 2. Spell Checker: Use your hashtable of Problem 1 to implement a simple spell checker. Build a dictionary (with the valid word as the key) using the words in the file 'small.dict'. After building your dictionary you will prompt the user to enter a word and print if it is a valid word or not. Rerun your program using the larger ispell dictionary file 'ispell.dict'

Since the Component-Sum is a bad hash-code map for strings, use Polynomial Accumulation to compute the hash-code map here. Compute the value of the polynomial using Horner's Rule and taking x = 33. Let the compression map remain unchanged from Problem 1.

You will need to be able to read in files: Search online on how to read in a file in Python

Problem 3. Auto-Correct Suggestions: Given an incorrectly spelled word, suggest a list of valid words from the dictionary that can be obtained by replacing exactly *one* character in the original word. For e.g. for the misspelt word 'tird', your program should print ['bird', 'gird', 'tied', 'tire']. Make sure you use the ispell dictionary for this problem.

Enhance your code to suggest valid words obtained by a single insert/delete operation on the misspelt word.

Problem 4*. Solve the previous problems by using a hash table that uses open addressing. Assume an initial table size of 8 (like in Python's hash table implementation) and double the table size whenever the load factor exceeds 0.75. Solve the problems using Linear Probing, Quadratic probing and Double Hashing. Can you find a way to compare their performances?

Problem 5*. Implement the O(1) `delete` operation for Problem 1 by using doubly linked lists.

* Problems 4 and 5 are optional.