Documentation

- ***Github link:***
  https://github.com/iuliaafarcas/FLCD/tree/main/Lab3
- ***Task:***

Statement: Implement the Symbol Table (ST) as the specified data structure, with the corresponding operations

Deliverables: class ST(source code) + documentation.

UPLOAD documentation, on the first line link to git for source code

Symbol Table (you need to implement the data structure and required operations) :

d. hash table

- ***Documentation:***

class SymbolTable:

- Description: It is a class containing an empty list of given capacity, filled with “buckets”. The buckets are lists that can contain elements placed after applying a hash function.

def hashFunction(self, word):

Arguments: - word: string; the value we are going to hash
Description: the function will establish the key where that word should be positioned
Return value: _sum % self.hashNumber where _sum is the ascii sum of all the letters in that word and hashNumber is an integer initialized in the constructor

def __init__(self):

Arguments: - capacity: integer; representing the capacity of the hash table; hashtable: dictionary where we are going to store the elements; hashnumber- integer: the number we are going to use for hashing
Description: the function sets the value for the capacity, hashnumber and will create the dictionary
Return value: -

def add(self, word):

Arguments: word: string. This is the value we are going to add in the hashtable
Description: the function compute the hashvalue of a given input and then tries to add it in the hashtable. If the key returned by the hashfunction if not in the set of keys already, then we will add it and then we will try to add the given value
Execution stops if the word is already in the hashtable.
Return value: -

def search(self, word):

Arguments: word: string. This is the value we are going to be searched in the hashtable
Description: we start from the first position in the hashtable trying to find the given value. We will use the following formula : i * self.hashNumber + key, where key is the value returned by the hashfunction, hashNumber is a determined number (we find it in the constructor) and i is the position of that word in the bucket
Return value: -1 if not found, i * self.hashNumber + key otherwise

def getValue(self, word):

Arguments: word: string
Description: we are searching for the given value
Return value: word if found in hashtable, none otherwise

def getOrAdd(self, word):

Arguments: word: string
Description: we are searching for the position of the given value. If found, return the position, if not, add the word and return its new position
Return value: integer