EECE 230X – Introduction to Computation and Programming
Programming Assignment 9

- This programming assignment consists of 4 problems
- Prerequisites: Topic 11
- Related material: Data structures: lists of lists, 2-dimensional lists, dictionaries, and stacks

Problem 1. Check if matrix has equal elements

Write a function `equalElements(M)`, which given an $m \times n$ matrix M of numbers, checks whether or not M has equal elements. That is, the function checks whether or not there are $(i_1, j_1) \neq (i_2, j_2)$, where $0 \leq i_1 < m$, $0 \leq j_1 < n$, $0 \leq i_2 < m$, and $0 \leq j_2 < n$, such that $M[i_1][j_1] = M[i_2][j_2]$. If so, the function should return the tuple of tuples $((i_1, j_1), (i_2, j_2))$, for any such (i_1, j_1) and (i_2, j_2). Otherwise, the function should return a tuple of tuples $((-1, -1), (-1, -1))$. If the matrix has more than two equal elements, the indices of any pair of equal elements are a valid answer.

a) $O(m^2n^2)$ **time solution.** Do it first using 4 nested loops on i_1, j_1, i_2, and j_2.

b) $O(mn)$ **expected time solution using a dictionary.** Do it now in $O(mn)$ expected time using a dictionary.

Don't try to read the following hint unless you are stuck (to read it you need to magnify the PDF file and properly read it in reverse order); try first to solve it on your own without help.

(*Hint:* .(j, i) selput era seulav eht dna $[j, i]M$ era syek yranoitcid ehT)

Test program:

```
import numpy as np
M1 = [[1,2],[3,4]]
M2 = [[1,2],[3,1]]
M3 = [[1,3,0,5],[2,5,2,-1],[5,6,-2,6]]
M4 = [[1,3,0,5],[20,50,2,-1],[51,61,-2,16]]
for M in (M1,M2,M3,M4):
    print(np.matrix(M))
    print(equalElements(M),"\n")
```

Output:

```
[[1 2]
 [3 4]]
((-1, -1), (-1, -1))

[[ 1 2]
 [3  1]]
((0, 0), (1, 1))

[[ 1  3  0  5]
 [ 2  5  2 -1]
 [ 5  6 -2  6]]
```

```
((0, 3), (1, 1))

[[ 1  3  0  5]
 [20 50  2 -1]
 [51 61 -2 16]]
((-1, -1), (-1, -1))
```

Problem 2. Check if two strings are anagrams using dictionaries

Two strings `s1` and `s2` are called *anagrams* if `s2` can be formed by rearranging the characters of `s1`.

Examples:

- `"3EE02CEC"` and `"EECE230C"` are anagrams
- `"EECE230C"` and `"EECE230C"` are anagrams
- `"3EEE02CE"` and `"EECE230C"` are not anagrams since `"C"` appears once in the first string and twice is in the second (and `"E"` appears 4 times in the first string and 3 times is in the second).
- `"aaabaab"` and `"baabaaa"` are anagrams
- `"aaabaab"` and `"abba"` are not anagrams

Write a function `anagrams(s1,s2)`, which given two strings `s1` and `s2`, returns `True` if they are anagrams, and `False` otherwise.

Aim for $O(n_1 + n_2)$ expected time, where $n_1 = len(s1)$ and $n_2 = len(s2)$. Use dictionaries.

(*Hint:* The operator `D1==D2` checks if dictionaries `D1` and `D2` have equal key:value pairs.)

Test program/Output:

```
print(anagrams("",""))                        True
print(anagrams("i","i"))                      True
print(anagrams("is","si"))                    True
print(anagrams("fun","nfu"))                  True
print(anagrams("aaabaab","abba"))             False
print(anagrams("aaabaab","baabaaa"))          True
print(anagrams("EECE230","EECE230"))          True
print(anagrams("EECE230","3EE02CE"))          True
print(anagrams("EECE230","3EEE02E"))          False
```

Problem 3. Longest zero sum sublist using a dictionary

Write a function `longestZeroSumSublist(L)`, which given a list `L` of integers, finds a (contiguous) sublist of `L` whose sum is zero and whose length is maximal.

If the list does not have a non-empty sublist whose elements sum to zero, your function should return empty list.

Examples: In each of the following examples, a zero-sum sublist of maximal length is underlined.

1 10 $\underline{\text{-1 -1 2}}$ 3 -5 26
1 10 $\underline{\text{-1 -1 4 3 -5}}$ 26
1 10 $\underline{\text{1 -1}}$ 4 3 -5 26
1 10 1 $\underline{\text{0}}$ 4 3 -5 26
1 10 1 1 4 3 -5 26
$\underline{\text{-1 -1 2}}$ 3 -5 26
2 2 $\underline{\text{-1 0 -1 2}}$
1 $\underline{\text{0 -2 1 0 1 -1 0 -1 2}}$ -2 -2

Note also that there are possibly more than one zero-sum sublist of maximal length, e.g., in the first example, we have two zero-sum sublists of maximal length: $1\ 10\ \underline{\mathbf{-1\ -1\ 2}}\ 3\ -5\ 26$ and $1\ 10\ -1\ -1\ \underline{\mathbf{2\ 3\ -5}}\ 26$. You are not asked to find all zero-sum sublists of maximal length; any one of them is a valid answer.

Below is the naive solution of this problem, which takes $O(n^2)$ steps, where n is the length of `L`.

```
def longestZeroSumSubListSlow(L): # O(n^2)
    print(L)
    n = len(L)
    if n== 0: return []
    iMax = 0
    jMax = -1
    for i in range(n):
        cumulativeSum = 0
        for j in range(i,n):
            cumulativeSum+= L[j]
            if  cumulativeSum==0 and  j - i > jMax-iMax:
                iMax = i
                jMax = j
    return L[iMax:jMax+1]
```

You are asked to do it in $O(n)$ expected time using a dictionary.

Don't try to read the following hint unless you are stuck (to read it you need to magnify the PDF file); try first to solve it on your own without help.

(*Hint :* Keep track of the cumulative sum. Let $s_i = L[0] + \ldots + L[i]$. Thus $L[i+1] + \ldots + L[j] = s_j - s_i$, hence $L[i+1] + \ldots + L[j] = 0$ if and only if $s_j = s_i$ Therefore, the length of the longest zero-sum sublist ending at j is $j - i$, where i is the first index such that $s_j = s_i$. How do we find i? Use dictionary D whose keys are cumulative sums. The value associated with key s is the index of the first occurrence of the cumulative sum s.)

```
 Test program:                                                        | Output:
                                                                      |
print(longestZeroSumSubList([1, 10, -1, -1, 2, 3, -5, 26]))           | [-1, -1, 2]
print(longestZeroSumSubList([1 ,10, -1, -1, 4, 3, -5, 26]))           | [-1, -1, 4, 3, -5]
print(longestZeroSumSubList([1, 10, 1, -1, 4, 3, -5, 26]))            | [1, -1]
print(longestZeroSumSubList([1, 10, 1, 0, 4, 3, -5, 26]))             | [0]
print(longestZeroSumSubList([1, 10, 1, 1, 4, 3, -5, 26]))             | []
print(longestZeroSumSubList([-1, -1, 2, 3, -5, 26]))                  | [-1, -1, 2]
print(longestZeroSumSubList([2, 2, -1, 0, -1, 2]))                    | [2, -1, 0, -1]
print(longestZeroSumSubList([1, 0, -2, 1, 0, 1, -1, 0, -1, 2, -2, -2])) | [0, -2, 1, 0, 1, -1,
                                                                      |  0, -1, 2]
```

Problem 4. Application of stacks: parentheses and braces checker

Write a function `parenthesesAndBracesChecker(s)`, which given a string `s` checks if the parentheses `"("  ")"` and braces `"["  "]"` match.

For example the parenthesis and braces match in `"a(aa)aa"`, `"aa(b(cd))e[ab]"`, and `"([aa(b)c[[aaaaa]]r(d)])"`, but they don't match in `"a([b)]"`, `"((aab)d"`, `"(("`, or `"ef)]"`.

Test your function on the above examples. Aim for $O(n)$ time, where $n = len(s)$.

(*Hint 1:* Use a stack.)

Don't try to read the following hint unless you are stuck (to read it you need to magnify the PDF file and properly read it in reverse order); try first to solve it on your own without help.

(*Hint 2:* gnirts eht ni sretcarahc eht htiw enod nehW ..... ,lobmys ”]“ ro ”)“ a ees uoy fI .ti hsup ,lobmys ”[“ ro ”(“ a ees uoy fI .”]“ dna ,”[“ ,”)“ ,”(“ morf tnereffid slobmys eht erongI .gnirts eht nacS)