Dict Problems

771. Jewels and Stones (https://leetcode.com/problems/jewels-and-stones/)

Easy

You're given strings J representing the types of stones that are jewels, and S representing the stones you have. Each character in S is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in J are guaranteed distinct, and all characters in J and S are letters. Letters are case sensitive, so "a" is considered a different type of stone from "A".

Example 1:

```
Input: J = "aA", S = "aAAbbbb"
Output: 3
```

Example 2:

```
Input: J = "z", S = "ZZ"
Output: 0
```

Note:

S and J will consist of letters and have length at most 50. The characters in J are distinct.

Performance

- Runtime: 20 ms, faster than 100.00% of Python online submissions for Jewels and Stones.
- Memory Usage: 10.7 MB, less than 100.00% of Python online submissions for Jewels and Stones.

Complexity Analysis

```
O(n + m) in time.
O(1)     in space.
```

In [1]:
```
from collections import defaultdict

class Solution(object):
    def numJewelsInStones(self, J, S):
        """
        :type J: str
        :type S: str
        :rtype: int
        """
        jewel_dict = defaultdict(bool)
        for j in J:
            jewel_dict[j] = True

        num_jewels = 0
        for s in S:
            if jewel_dict[s]:
                num_jewels += 1

        return num_jewels

my_sol = Solution()
print("Should print 3:", my_sol.numJewelsInStones("aA", "aAAbbbb"))
print("Should print 0:", my_sol.numJewelsInStones("z", "ZZ"))
```

```
Should print 3: 3
Should print 0: 0
```

804. Unique Morse Code Words (https://leetcode.com/problems/unique-morse-code-words/)

Easy

International Morse Code defines a standard encoding where each letter is mapped to a series of dots and dashes, as follows: "a" maps to ".-", "b" maps to "-...", "c" maps to "-.-.", and so on.

For convenience, the full table for the 26 letters of the English alphabet is given below:

```
[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--
",
 "-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","-
--.."]
```

Now, given a list of words, each word can be written as a concatenation of the Morse code of each letter. For example, "cba" can be written as "-.-..--...", (which is the concatenation "-.-." + "-..." + ".-"). We'll call such a concatenation, the transformation of a word.

Return the number of different transformations among all words we have.

```
Example:
Input: words = ["gin", "zen", "gig", "msg"]

Output: 2
```

Explanation: The transformation of each word is:

```
"gin" -> "--...-."
"zen" -> "--...-."
"gig" -> "--.--."
"msg" -> "--.--."
```

There are 2 different transformations, "--...-." and "--...--.".

Note:

- The length of words will be at most 100.
- Each words[i] will have length in range [1, 12].
- words[i] will only consist of lowercase letters.

Performance

- Runtime: 40 ms, faster than 64.65% of Python3 online submissions for Unique Morse Code Words.
- Memory Usage: 13.1 MB, less than 5.36% of Python3 online submissions for Unique Morse Code Words.

Complexity Analysis

- O(n) in time.
- O(n) in space (at worst there are n morse codes I store in the morse_dict).

In [2]:
```
from collections import defaultdict

c_to_m = [".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--
          "-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--

class Solution:
    def uniqueMorseRepresentations(self, words: 'List[str]') -> 'int':
        morse_dict = defaultdict(bool)
        n_codes = 0
        for word in words:
            word = word.lower()
            morse = ''.join([ c_to_m[ord(letter)-ord('a')] for letter in word ])
            if not morse_dict[morse]:
                n_codes += 1
            morse_dict[morse] = True
        return n_codes

my_sol = Solution()
print('Should print 2:', my_sol.uniqueMorseRepresentations(["gin", "zen", "gig", "msg"]
```

```
Should print 2: 2
```

290. Word Pattern (https://leetcode.com/problems/word-pattern/)

Easy

Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.

Example 1:

```
Input: pattern = "abba", str = "dog cat cat dog"
Output: true
```

Example 2:

```
Input: pattern = "abba", str = "dog cat cat fish"
Output: false
```

Example 3:

```
Input: pattern = "aaaa", str = "dog cat cat dog"
Output: false
```

Example 4:

```
Input: pattern = "abba", str = "dog dog dog dog"
Output: false
```

Notes:

You may assume pattern contains only lowercase letters, and str contains lowercase letters separated by a single space.

Performance

- Runtime: 32 ms, faster than 99.90% of Python3 online submissions for Word Pattern.
- Memory Usage: 12.3 MB, less than 100.00% of Python3 online submissions for Word Pattern.

Complexity Analysis

```
O(n) in time.
O(n) in space.
```

In [3]:
```python
class Solution:
    def wordPattern(self, pattern: 'str', string: 'str') -> 'bool':
        words = string.split()
        if len(pattern) != len(words):
            return False
        p_dict = {}
        w_dict = {}
        for i, p in enumerate(pattern):
            if p in p_dict and p_dict[p] != words[i]:
                return False
            if not(p in p_dict):
                if words[i] in w_dict:
                    return False
                else:
                    p_dict[p] = words[i]
                    w_dict[words[i]] = True
        return True

my_sol = Solution()
print('Should print True:', my_sol.wordPattern('abba', 'dog cat cat dog'))
print('Should print False:', my_sol.wordPattern('abba', 'dog cat cat fish'))
print('Should print False:', my_sol.wordPattern('aaaa', 'dog cat cat dog'))
print('Should print False:', my_sol.wordPattern('abba', 'dog dog dog dog'))
```

```
Should print True: True
Should print False: False
Should print False: False
Should print False: False
```

389. Find the Difference (https://leetcode.com/problems/find-the-difference/)

Easy

Given two strings s and t which consist of only lowercase letters.

String t is generated by random shuffling string s and then add one more letter at a random position.

Find the letter that was added in t.

Example:

```
Input:
s = "abcd"
t = "abcde"

Output:
e
```

Explanation:

'e' is the letter that was added.

Performance

- Runtime: 36 ms, faster than 99.62% of Python3 online submissions for Find the Difference.
- Memory Usage: 12.5 MB, less than 100.00% of Python3 online submissions for Find the Difference.

Complexity Analysis

```
O(n) in time
O(n) in space
```

In [4]:
```
from collections import Counter

class Solution:
    def findTheDifference(self, s: 'str', t: 'str') -> 'str':
        s_count = Counter(s)
        t_count = Counter(t)
        for k, v in t_count.items():
            if not(k in s_count):
                return k
            else:
                if v != s_count[k]:
                    return k

my_sol = Solution()
print('Should print e:', my_sol.findTheDifference('abcd', 'abcde'))
```

```
Should print e: e
```

347. Top K Frequent Elements (https://leetcode.com/problems/top-k-frequent-elements/)

Medium

Given a non-empty array of integers, return the k most frequent elements.

Example 1:

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

Example 2:

```
Input: nums = [1], k = 1
Output: [1]
```

Note:

- You may assume k is always valid, 1 ≤ k ≤ number of unique elements.
- Your algorithm's time complexity must be better than O(n log n), where n is the array's size.

Performance

- Runtime: 44 ms, faster than 99.93% of Python3 online submissions for Top K Frequent Elements.
- Memory Usage: 15.4 MB, less than 100.00% of Python3 online submissions for Top K Frequent Elements.

Complexity Analysis

```
O(n log n) in time
O(n) in space
```

In [5]:
```
from collections import Counter

class Solution:
    def topKFrequent(self, nums: 'List[int]', k: 'int') -> 'List[int]':
        count = Counter(nums)
        l_count = [(v, k) for k, v in count.items()]
        l_count.sort(reverse=True)
        return [k for (v, k) in l_count[:min(k, len(l_count))]]

my_sol = Solution()
print('Should print [1, 2]:', my_sol.topKFrequent([1,1,1,2,2,3], 2))
```

```
Should print [1, 2]: [1, 2]
```

Coding Mock Interview (3.30pm 4 Apr 2019)

Given two dictionaries find the differences, if there are none return None

In [6]:
```
def diff(arg1, arg2):
    """return first difference between the args if there is one,
    else return None"""

    # check if the types of arg1 and arg2 are the same
    if type(arg1) != type(arg2):
        return arg1, arg2

    # if args aren't a list or a dict just compare
    if type(arg1) != dict and type(arg1) != list:
        if arg1 != arg2:
            return arg1, arg2

    # if type args are lists
    if type(arg1) == list:
        for idx, elt in ennumerate(arg1):
            if diff(elt, arg2[idx]) != None:
                return diff(elt, arg2[idx])

    # if type args are dicts
    if type(arg1) == dict:
        if diff(list(arg1.keys()).sort(), list(arg2.keys()).sort()) != None:
            return diff(list(arg1.keys()).sort(), list(arg2.keys()).sort())
        for k, v in arg1.items():
            if arg2[k] != v:
                return arg1, arg2
    return None


my_dict = {'a':1, 'b':2}
this_dict = {'c':2, 'd':3}
print(my_dict==this_dict)
```

```
False
```

Coderpad Interview (6pm 25 Feb 2019)

- You're writing a ransom note by clipping words out of a body of text (`anna_karenina.txt`).
- Write efficient code to check whether a particular note can be clipped from the input text.

In [ ]:
```
textFileString = 'a b c d e f'

We note = 'Place $100,000 cash in a bag outside Liverpool Street Station at noon tomorr

def checkWordsInFile(note):
    listOfWords = note.split(' ')
    print(listOfWords)

    #file = open('anna_karena.txt',mode='r')
    #fileString = file.read()
    #file.close()

    file_string = textFileString
    words_in_txt_file = file_string.split(' ')

    for word in listOfWords:
        if not(word in words_in_text_file)
```

383. Ransom Note (https://leetcode.com/problems/ransom-

note/)

Easy

Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

Note: You may assume that both strings contain only lowercase letters.

```
canConstruct("a", "b") -> false
canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true
```

In [ ]:
```
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: