set7.py

```python
#1 Valid Anagram

#Method:
    # Use a Hash Map: Count the frequency of each character in string s. Decrease the counts
    # while scanning string t. If all counts return to zero, then the strings are anagrams.

#Constraints: Assume the strings consist of lowercase English letters.

class HashTable:
    def __init__(self):
        self.arr = []
        for i in range(26):
            self.arr.append(0)

    #hash_function(char) = (ord(char) - ord('a')) % 26
    def insert(self,char):
        idx = (ord(char) - ord('a')) % 26
        self.arr[idx] +=1

    def erase(self,char):
        idx = (ord(char) - ord('a')) % 26
        self.arr[idx] -=1

    def isempty(self):
        for i in self.arr:
            if(i != 0):
                return False
        return True


def validAnagram(s , t):
    map = HashTable()
    for i in s:
        map.insert(i)

    for j in t:
        map.erase(j)

    if(map.isempty()):
        return True
    return False

print(validAnagram("anagram", "nagaram"))
print(validAnagram("rat","car"))


#collision handling map
class HashMap:
```

```
49      def __init__(self,s):
50          self.size = len(s)
51          self.keys = [None]*self.size
52          self.values = [0]*self.size
53          self.occupied = [False]*self.size
54
55      def hash_func(self, c):
56          return (ord(c) - ord('a'))%self.size
57
58      def insert(self, c):
59          idx = self.hash_func(c)
60          start_idx = idx
61          while(self.occupied[idx] == True and self.keys[idx] != c):
62              idx = (idx+1)%self.size
63              if(idx == start_idx):
64                  print("HashTable is full")
65                  return
66
67          if(self.occupied[idx] == False):
68              self.occupied[idx] = True
69              self.keys[idx] = c
70              self.values[idx] = 1
71          else:
72              self.values[idx] +=1
73
74      def decrement(self, c):
75          idx = self.hash_func(c)
76          start_idx = idx
77          while(self.occupied[idx] == True):
78              if(self.keys[idx] == c):
79                  self.values[idx] -=1
80                  break
81              idx = (idx + 1)%self.size
82              if(idx == start_idx):
83                  break
84
85      def all_zero(self):
86          for i in range(self.size):
87              if(self.occupied[i] == True):
88                  if(self.values[i] != 0):
89                      return False
90          return True
91
92
93
94  def is_anagram(s, t):
95      if(len(s) != len(t)):
96          return False
97      map = HashMap(s)
98      for i in s:
```

```
        map.insert(i)

    for j in t:
        map.decrement(j)

    return map.all_zero()


print(is_anagram("anagram", "nagaram"))
print(is_anagram("rat", "car"))


#2 Group Anagrams
class Node:
    def __init__(self,key,value):
        self.key = key
        self.value = [value]

class HashMap:
    def __init__(self,words):
        self.size = len(words)
        self.arr = [None]*self.size

    def hash_function(self,word):
        idx = 0
        for char in word:
            idx = (idx*31 + (ord(char)-ord('a')))%self.size
        return idx

    def insert(self, word):
        idx = self.hash_function(word)
        key = ''.join(sorted(word))
        start_idx = idx

        #search
        while(self.arr[idx] != None and self.arr[idx].key != key):
            idx = (idx+1)%self.size
            if(idx == start_idx):
                print("Map is Full")
                return

        #got it
        if(self.arr[idx] == None):
            self.arr[idx] = Node(key,word)
        else:
            self.arr[idx].value.append(word)

    def search(self,word):
        idx = self.hash_function(word)
        key = ''.join(sorted(word))
```

```
149         start_idx = idx
150 
151         while(self.arr[idx] != None):
152             if(self.arr[idx].key == key):
153                 return True
154             idx = (idx+1)%self.size
155             if(idx == start_idx):
156                 return False
157         return False
158 
159     def display(self):
160         all_words = []
161         for words in self.arr:
162             if words != None:
163                 print(words.value)
164                 all_words.append(words.value)
165         print(all_words)
166 
167 words = ["eat", "tea", "tan", "ate", "nat", "bat"]
168 h = HashMap(words)
169 for word in words:
170     h.insert(word)
171 
172 h.display()
```