TASK 1:

1) Count the occurrences of each letter in the text.

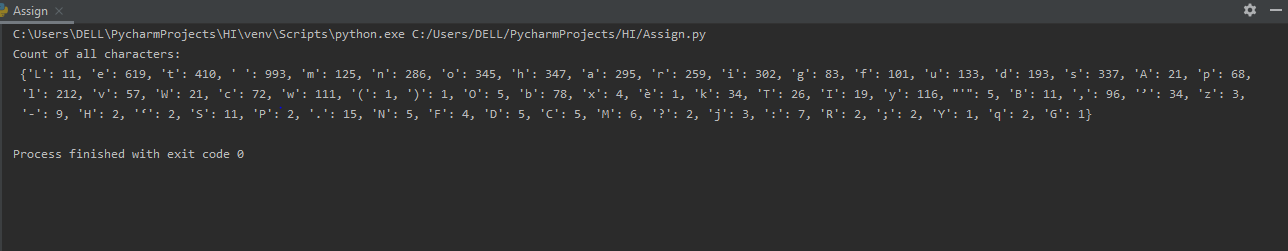
Algorithm:

1. Create an empty dictionary.
2. Update values when it finds the same character

Code:

freq = {}  
# A  
for i in teststr:  
 if i in freq:  
 freq[i] += 1  
 else:  
 freq[i] = 1

Output:



2) Print the number of one-letter, two-letter, three-letter words and so on.

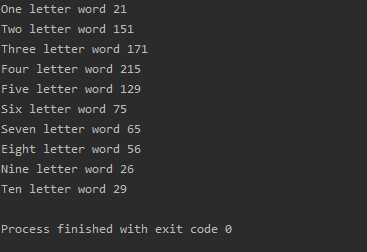
Algorithm:

1. Let counts of each letter word.
2. Initialize them with zero
3. Check lengths of words and update according to it

Code:

count1 = 0;  
count2 = 0;  
count3 = 0;  
count4 = 0;  
count5 = 0;  
count6 = 0;  
count7 = 0;  
count8 = 0;  
count9 = 0;  
count10 = 0;  
for j in data:  
 if len(j) == 1:  
 count1 = count1 + 1;  
 elif len(j) == 2:  
 count2 = count2 + 1;  
 elif len(j) == 3:  
 count3 = count3 + 1;  
 elif len(j) == 4:  
 count4 = count4 + 1;  
 elif len(j) == 5:  
 count5 = count5 + 1;  
 elif len(j) == 6:  
 count6 = count6 + 1;  
 elif len(j) == 7:  
 count7 = count7 + 1;  
 elif len(j) == 8:  
 count8 = count8 + 1;  
 elif len(j) == 9:  
 count9 = count9 + 1;  
 elif len(j) == 10:  
 count10 = count10 + 1;  
  
print("One letter word " + str(count1))  
print("Two letter word " + str(count2))  
print("Three letter word " + str(count3))  
print("Four letter word " + str(count4))  
print("Five letter word " + str(count5))  
print("Six letter word " + str(count6))  
print("Seven letter word " + str(count7))  
print("Eight letter word " + str(count8))  
print("Nine letter word " + str(count9))  
print("Ten letter word " + str(count10))

Output :



3) Print the number of occurrences of each different word in the text.

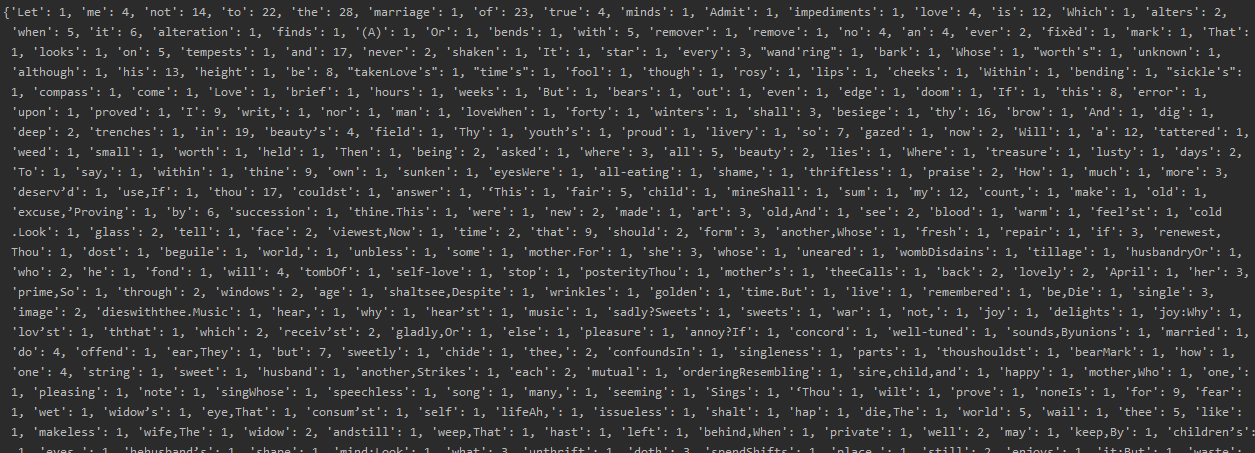
Algorithm:

1. Create an empty tuple
2. Count words and if word is found increase the count

Code:

counts = dict()  
for word in data:  
 if word in counts:  
 counts[word] += 1  
 else:  
 counts[word] = 1  
  
print(counts)

Output:



{'Let': 1, 'me': 4, 'not': 14, 'to': 22, 'the': 28, 'marriage': 1, 'of': 23, 'true': 4, 'minds': 1, 'Admit': 1, 'impediments': 1, 'love': 4, 'is': 12, 'Which': 1, 'alters': 2, 'when': 5, 'it': 6, 'alteration': 1, 'finds': 1, '(A)': 1, 'Or': 1, 'bends': 1, 'with': 5, 'remover': 1, 'remove': 1, 'no': 4, 'an': 4, 'ever': 2, 'fixèd': 1, 'mark': 1, 'That': 1, 'looks': 1, 'on': 5, 'tempests': 1, 'and': 17, 'never': 2, 'shaken': 1, 'It': 1, 'star': 1, 'every': 3, "wand'ring": 1, 'bark': 1, 'Whose': 1, "worth's": 1, 'unknown': 1, 'although': 1, 'his': 13, 'height': 1, 'be': 8, "takenLove's": 1, "time's": 1, 'fool': 1, 'though': 1, 'rosy': 1, 'lips': 1, 'cheeks': 1, 'Within': 1, 'bending': 1, "sickle's": 1, 'compass': 1, 'come': 1, 'Love': 1, 'brief': 1, 'hours': 1, 'weeks': 1, 'But': 1, 'bears': 1, 'out': 1, 'even': 1, 'edge': 1, 'doom': 1, 'If': 1, 'this': 8, 'error': 1, 'upon': 1, 'proved': 1, 'I': 9, 'writ,': 1, 'nor': 1, 'man': 1, 'loveWhen': 1, 'forty': 1, 'winters': 1, 'shall': 3, 'besiege': 1, 'thy': 16, 'brow': 1, 'And': 1, 'dig': 1, 'deep': 2, 'trenches': 1, 'in': 19, 'beauty’s': 4, 'field': 1, 'Thy': 1, 'youth’s': 1, 'proud': 1, 'livery': 1, 'so': 7, 'gazed': 1, 'now': 2, 'Will': 1, 'a': 12, 'tattered': 1, 'weed': 1, 'small': 1, 'worth': 1, 'held': 1, 'Then': 1, 'being': 2, 'asked': 1, 'where': 3, 'all': 5, 'beauty': 2, 'lies': 1, 'Where': 1, 'treasure': 1, 'lusty': 1, 'days': 2, 'To': 1, 'say,': 1, 'within': 1, 'thine': 9, 'own': 1, 'sunken': 1, 'eyesWere': 1, 'all-eating': 1, 'shame,': 1, 'thriftless': 1, 'praise': 2, 'How': 1, 'much': 1, 'more': 3, 'deserv’d': 1, 'use,If': 1, 'thou': 17, 'couldst': 1, 'answer': 1, '‘This': 1, 'fair': 5, 'child': 1, 'mineShall': 1, 'sum': 1, 'my': 12, 'count,': 1, 'make': 1, 'old': 1, 'excuse,’Proving': 1, 'by': 6, 'succession': 1, 'thine.This': 1, 'were': 1, 'new': 2, 'made': 1, 'art': 3, 'old,And': 1, 'see': 2, 'blood': 1, 'warm': 1, 'feel’st': 1, 'cold.Look': 1, 'glass': 2, 'tell': 1, 'face': 2, 'viewest,Now': 1, 'time': 2, 'that': 9, 'should': 2, 'form': 3, 'another,Whose': 1, 'fresh': 1, 'repair': 1, 'if': 3, 'renewest,Thou': 1, 'dost': 1, 'beguile': 1, 'world,': 1, 'unbless': 1, 'some': 1, 'mother.For': 1, 'she': 3, 'whose': 1, 'uneared': 1, 'wombDisdains': 1, 'tillage': 1, 'husbandryOr': 1, 'who': 2, 'he': 1, 'fond': 1, 'will': 4, 'tombOf': 1, 'self-love': 1, 'stop': 1, 'posterityThou': 1, 'mother’s': 1, 'theeCalls': 1, 'back': 2, 'lovely': 2, 'April': 1, 'her': 3, 'prime,So': 1, 'through': 2, 'windows': 2, 'age': 1, 'shaltsee,Despite': 1, 'wrinkles': 1, 'golden': 1, 'time.But': 1, 'live': 1, 'remembered': 1, 'be,Die': 1, 'single': 3, 'image': 2, 'dieswiththee.Music': 1, 'hear,': 1, 'why': 1, 'hear’st': 1, 'music': 1, 'sadly?Sweets': 1, 'sweets': 1, 'war': 1, 'not,': 1, 'joy': 1, 'delights': 1, 'joy:Why': 1, 'lov’st': 1, 'ththat': 1, 'which': 2, 'receiv’st': 2, 'gladly,Or': 1, 'else': 1, 'pleasure': 1, 'annoy?If': 1, 'concord': 1, 'well-tuned': 1, 'sounds,Byunions': 1, 'married': 1, 'do': 4, 'offend': 1, 'ear,They': 1, 'but': 7, 'sweetly': 1, 'chide': 1, 'thee,': 2, 'confoundsIn': 1, 'singleness': 1, 'parts': 1, 'thoushouldst': 1, 'bearMark': 1, 'how': 1, 'one': 4, 'string': 1, 'sweet': 1, 'husband': 1, 'another,Strikes': 1, 'each': 2, 'mutual': 1, 'orderingResembling': 1, 'sire,child,and': 1, 'happy': 1, 'mother,Who': 1, 'one,': 1, 'pleasing': 1, 'note': 1, 'singWhose': 1, 'speechless': 1, 'song': 1, 'many,': 1, 'seeming': 1, 'Sings': 1, '‘Thou': 1, 'wilt': 1, 'prove': 1, 'noneIs': 1, 'for': 9, 'fear': 1, 'wet': 1, 'widow’s': 1, 'eye,That': 1, 'consum’st': 1, 'self': 1, 'lifeAh,': 1, 'issueless': 1, 'shalt': 1, 'hap': 1, 'die,The': 1, 'world': 5, 'wail': 1, 'thee': 5, 'like': 1, 'makeless': 1, 'wife,The': 1, 'widow': 2, 'andstill': 1, 'weep,That': 1, 'hast': 1, 'left': 1, 'behind,When': 1, 'private': 1, 'well': 2, 'may': 1, 'keep,By': 1, 'children’s': 1, 'eyes,': 1, 'hehusband’s': 1, 'shape': 1, 'mind:Look': 1, 'what': 3, 'unthrift': 1, 'doth': 3, 'spendShifts': 1, 'place,': 1, 'still': 2, 'enjoys': 1, 'it;But': 1, 'waste': 1, 'hath': 5, 'end,And': 1, 'kept': 1, 'unused'

Task 2:

Find the fewest words that will link them.

Algorithm:

1. Create empty list of lists to store all possible paths
2. ‘combine ’ is a separate list used to store transformed words
3. ‘queue’ is used for implementation of BFS
4. It checks if the word length is same as of start word and only one word differ so it adds in queue.
5. If it reaches the end word the whole queue is shifted to paths
6. Queue is empty again and process continues

Code:

:  
  
 Lists = open('Words.txt').read().splitlines()  
  
start1 = "flour"  
target1 = "bread"  
start2 = "chaos"  
target2 = "peace"  
start3 = "river"  
target3 = "shore"  
start4 = "sleep"  
target4 = "dream"  
start5 = "black"  
target5 = "white"  
start6 = "witch"  
target6 = "fairy"  
start7 = "tears"  
target7 = "smile"  
start8 = "which"  
target8 = "think"  
start9 = "paper"  
target9 = "story"  
start10 = "early"  
target10 = "trees"  
  
  
  
  
  
def minladder( beginWord: str, endWord: str, wordList: List[str]) -> List[List[str]]:  
 combine = collections.defaultdict(list)  
 x = len(beginWord)  
 for word in wordList:  
 for i in range(x):  
 combine[word[0:i] + "\*" + word[i + 1:]].append(word)  
  
  
 queue, paths, dis = collections.deque([(beginWord, [beginWord])]), [], {}  
 while queue:  
 word, path = queue.popleft()  
 for i in range(x):  
 key = word[0:i] + "\*" + word[i + 1:]  
 for w in combine[key]:  
 # avoid loop visit  
 if w in dis and dis[w] <= len(path):  
 continue  
 dis[w] = len(path) + 1  
   
 if w == endWord and (not paths or len(path) + 1 <= len(paths[-1])):  
 paths.append(path + [w])  
 break  
 queue.append((w, path + [w]))  
  
 if not paths:  
 return []  
 return [paths[0]]  
  
  
print("Flour and bread: " + str(minladder(start1, target1, Lists)))  
print("Chaos and peace: " + str(minladder(start2, target2, Lists)))  
print("River and shore: " + str(minladder(start3, target3, Lists)))  
print("Sleep and dream: " + str(minladder(start4, target4, Lists)))  
print("Black and white: " + str(minladder(start5, target5, Lists)))  
print("Witch and fairy: " + str(minladder(start6, target6, Lists)))  
print("tears and smile: " + str(minladder(start7, target7, Lists)))  
print("which and think: " + str(minladder(start8, target8, Lists)))  
print("paper and story: " + str(minladder(start9, target9, Lists)))  
print("early and trees: " + str(minladder(start10, target10, Lists)))

Output:

