(a.)
15 most frequent 5-letter words:

word	count
THREE	273077
SEVEN	178842
EIGHT	165764
WOULD	159875
ABOUT	157448
THEIR	145434
WHICH	142146
AFTER	110102
FIRST	109957
FIFTY	106869
OTHER	106052
FORTY	94951
YEARS	88900
THERE	86502
SIXTY	73086

14 least frequent 5-letter words:

word	count
BOSAK	6
CAIXA	6
MAPCO	6
OTTIS	6
TROUP	6
CCAIR	7
CLEFT	7
FABRI	7
FOAMY	7
NIAID	7
PAXON	7
SERNA	7
TOCOR	7
YALOM	7

(b.)

Correctly guessed	Incorrectly guessed	Best next guess letter	Probability
— — — — —	{}	E	0.5394
— — — — —	{E, A}	O	0.5340
A — — — S	{}	E	0.7715
A — — — S	{I}	E	0.7127
— — O — —	{A, E, M, N, T}	R	0.7454

(c.)

```
import pandas as pd
import numpy as np

inputData = pd.read_csv('hw1_word_counts_05.txt', sep=" ", header = None, names = ["word", "count"])
#print(inputData.head(10))
#print("Number of words from input: ", inputData.shape[0])

# Show the result of problem (a.)
inputData = inputData.sort_values(by = ["count","word"], ascending = False)
print("15 most frequent 5-letter words:")
print("==================")
print(inputData.head(15).to_string(index = False))
print("==================")
print("\n14 least frequent 5-letter words:")
print("==================")
print(inputData.tail(14).sort_values(by = ["count","word"]).to_string(index = False))
print("==================")

# Calculate P(W=w)
totalCounts = inputData["count"].sum()
print("Total counts: ", totalCounts)
inputData['prob'] = inputData['count'] / totalCounts
print(inputData.head().to_string(index = False))

#Reocrd the current state of correctly guessed chars and their position
charPositions = range(1,6)
correctChars = ["-" for i in range(1,6)]
correctEvids = pd.DataFrame({'Position': charPositions, 'Chars':correctChars})
```

```
print(correctEvids)

#Reocrd the current state of incorrectly guessed chars or chars used in other positions
incorrectChars = [set() for i in range(1,6)]
incorrectEvids = pd.DataFrame({'Position': charPositions, 'Chars':incorrectChars})
print(incorrectEvids)

# Sets complement of positive constraints in negative constraints
def set_incorrect_evidences(position, value):
    for index, row in incorrectEvids.iterrows():
        if row['Position'] != position:
            incorrectEvids.at[row['Position']-1, 'Chars'].add(value)

# Below show an example of calulating the 4th situation from problem (b.)
# Set up the current state of correct evidences
allGuessedChars = set({'A','S'})
correctEvids.at[0, 'Chars'] = "A"
correctEvids.at[4, 'Chars'] = "S"
#print(correctEvids)

# Set up the current state of incorrect evidences
for index, row in correctEvids.iterrows():
    if row['Chars'] != "-" and row['Chars'] != ' ':
        set_incorrect_evidences(row['Position'], row['Chars'])

# Set up incorrectly guessed chars to incorrect evidences
incorGuessedChars = []
incorGuessedChars.append('I')
if incorGuessedChars:
    allGuessedChars.update(incorGuessedChars)
    for i in incorGuessedChars:
        for index, row in incorrectEvids.iterrows():
            incorrectEvids.at[index, 'Chars'].add(i)

#print(incorrectEvids)

def word_match_correct_evidences(correctEvids, word):
    for index, row in correctEvids.iterrows():
        if row['Chars'] != '-':
            if word[index] != row['Chars']:
                return 0
    return 1
```

```
def word_match_incorrect_evidences(incorrectEvids, word):
    for index, row in incorrectEvids.iterrows():
        if row['Chars']:
            if word[index] in row['Chars']:
                return 0
    return 1

def prob_of_evid_given_word(correctEvids, incorrectEvids, word):
    return word_match_correct_evidences(correctEvids, word) and
word_match_incorrect_evidences(incorrectEvids, word)

letters =
["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","
Z"]
probLettersGivenEvids =  pd.DataFrame({'Letter':letters, 'Probability':np.zeros(26)})
print(probLettersGivenEvids)

for index, row in probLettersGivenEvids.iterrows():
    curLetter = row['Letter']
    # Skip those already guessed
    if curLetter in allGuessedChars:
        continue

    # Start calculating the probability for current letter
    probCurLetterAllWords = 0
    print("Calculating probability of letter: ", row['Letter'])

    # Handle Sigma( P(L=l for some i={1--5}|W=w) * P(W=w|E) )
    for inputDataIndex, inputDataRow in inputData.iterrows():
        curWord = inputDataRow['word']

        # Handle P(L=l for some i={1--5}|W=w), an 0-1 prob
        if curLetter not in curWord:
            probLettersGivenEvids.at[index, 'Probability'] = 0
            continue

        # Handle P(W=w|E), the posterior probability
        # numerator is P(E|W=w) * P(W=w)
        numerator = prob_of_evid_given_word(correctEvids, incorrectEvids, curWord) *
inputDataRow['prob']
        if numerator == 0:
            probLettersGivenEvids.at[index, 'Probability'] = 0
            continue
```

```
        # denominator is Sigma( P(E|W=w')*P(W=w') )
        denominator = 0
        for inputDataIndex2, inputDataRow2 in inputData.iterrows():
            denominator += prob_of_evid_given_word(correctEvids, incorrectEvids,
inputDataRow2['word']) * inputDataRow2['prob']

        # Sum into the probability of the current letter
        probCurLetterAllWords += numerator/denominator

    print("Probability for: ", row['Letter']," is :", probCurLetterAllWords)
    probLettersGivenEvids.at[index, 'Probability'] = probCurLetterAllWords

# Show the final result of letter with highest probability
print(probLettersGivenEvids.sort_values("Probability", ascending=False).head(1))
```