*****BRIEF EXPLANATIONS AT THE END ******

TASK 1: VOCABULARY CREATION

In [63]:
```python
from collections import defaultdict
import operator

# Reading the Training Data
file_path = r"C:\Users\adity\Desktop\NLP\CSCI544_HW3\hw3\data\train"
word_counts = defaultdict(int)

with open(file_path, 'r') as file:
    for line in file:
        if line.strip():  # skips blank lines
            _, word, _ = line.strip().split('\t')
            word_counts[word] += 1

#  Counting Word Occurrences and Replacing Rare Words
threshold = 3
vocabulary = {'<unk>': 0}
for word, count in word_counts.items():
    if count >= threshold:
        vocabulary[word] = count
    else:
        vocabulary['<unk>'] += count

#  Sorting and Indexing Vocabulary
sorted_vocab = sorted(vocabulary.items(), key=operator.itemgetter(1), reverse=True)
indexed_vocab = {word: (index, count) for index, (word, count) in enumerate(sorted_vocab)}

#  Write to Vocabulary File
with open('vocab.txt', 'w') as vocab_file:
    for word, (index, count) in indexed_vocab.items():
        vocab_file.write(f"{index}\t{word}\t{count}\n")

# Display size of vocabulary and total occurrences of '<unk>'
print(f"Total vocabulary size: {len(indexed_vocab)}")
print(f"Total occurrences of '<unk>': {indexed_vocab['<unk>'][1]}")
```

```
Total vocabulary size: 16920
Total occurrences of '<unk>': 32537
```

In [64]:
```python
# Printing first 10 lines of the vocab.txt file to check result
with open('vocab.txt', 'r') as vocab_file:
    lines = [next(vocab_file) for _ in range(10)]
    print(''.join(lines))
```

```
0       ,       46476
1       the     39533
2       .       37452
3       <unk>   32537
4       of      22104
5       to      21305
6       a       18469
7       and     15346
8       in      14609
9       's      8872
```

TASK 2: MODEL LEARNING

In [65]:
```python
import json
from collections import defaultdict, Counter


file_path = r"C:\\Users\\adity\\Desktop\\NLP\\CSCI544_HW3\\hw3\\data\\train"

# Initializing dictionaries
transition_counts = defaultdict(Counter)
emission_counts = defaultdict(Counter)
tag_counts = Counter()

# Reading training data
with open(file_path, 'r') as file:
    prev_tag = None  # To keep track of previous tag for transition counts
    for line in file:
        if line.strip():
            _, word, tag = line.strip().split('\t')
            emission_counts[tag][word] += 1
            tag_counts[tag] += 1
            if prev_tag is not None:  # If not start of the file
                transition_counts[prev_tag][tag] += 1
            prev_tag = tag
        else:  # Reset prev_tag at end
            prev_tag = None

# Calculating transition probabilities
transition_probs = {s: {s_prime: count / tag_counts[s] for s_prime, count in s_counts.items()}
                    for s, s_counts in transition_counts.items()}

# Calculating emission probabilities
emission_probs = {tag: {word: count / tag_counts[tag] for word, count in word_counts.items()}
                  for tag, word_counts in emission_counts.items()}

# Write probabilities to JSON file
hmm_model = {'transition': transition_probs, 'emission': emission_probs}
with open('hmm.json', 'w') as f:
    json.dump(hmm_model, f, indent=2)

# Output no. of transition and emission parameters
print(f"Number of transition parameters: {sum(len(s_counts) for s_counts in transition_probs.values())}")
print(f"Number of emission parameters: {sum(len(word_counts) for word_counts in emission_probs.values())}")
```

```
Number of transition parameters: 1351
Number of emission parameters: 50286
```

In [66]:
```python
# Printing small sample of transition probabilities
print("Sample transition probabilities:")
for tag, following_tags in list(transition_probs.items())[:5]:
    print(f"{tag}: {dict(list(following_tags.items())[:5])}")

# Printing small sample of emission probabilities
print("\nSample emission probabilities:")
for tag, words in list(emission_probs.items())[:5]:
    print(f"{tag}: {dict(list(words.items())[:5])}")
```

```
Sample transition probabilities:
NNP: {'NNP': 0.3782645420509543, ',': 0.13846908958086018, 'CD': 0.019176330928682313, 'VBZ': 0.0391973335768423, 'VBG': 0.0017692448178248561}
,: {'CD': 0.021234939759036144, 'MD': 0.010542168674698794, 'DT': 0.1336273666092943, 'VBD': 0.05154905335628227, 'NNS': 0.02732358003442341}
CD: {'NNS': 0.15775891730703062, '.': 0.0725427227893107, 'CC': 0.017175134763160915, 'TO': 0.037590319990824635, ',': 0.09548113315747218}
NNS: {'JJ': 0.017196978862406887, 'VBZ': 0.008520714149916175, 'IN': 0.2345183981748734, 'VBN': 0.020930192364195715, 'VBD': 0.07125944105497849}
JJ: {',': 0.029129343105320303, 'NN': 0.4491042345276873, 'CC': 0.01701615092290988, 'JJ': 0.07400244299674268, 'IN': 0.05652823018458197}

Sample emission probabilities:
NNP: {'Pierre': 6.84868961738654e-05, 'Vinken': 2.2828965391288468e-05, 'Nov.': 0.0026709889507807506, 'Mr.': 0.044014245274404167, 'Elsevier': 1.1414482695644234e-05}
,: {',': 0.9999139414802065, 'Wa': 2.151462994836489e-05, 'an': 2.151462994836489e-05, '2': 2.151462994836489e-05, 'underwriters': 2.151462994836489e-05}
CD: {'61': 0.0007168253240050465, '29': 0.0021218029590549374, '55': 0.0015483426998509004, '30': 0.013562335130175478, '1956': 8.601903888060558e-05}
NNS: {'years': 0.019530237301024905, 'filters': 0.00015555056257453463, 'deaths': 0.0005012184794068339, 'workers': 0.003404828980798147, 'researchers': 0.0011579875213882024}
JJ: {'old': 0.003613599348534202, 'nonexecutive': 0.00010179153094462541, 'former': 0.004377035830618893, 'British': 0.0032742942453854508, 'industrial': 0.002120656894679696}
```

TASK 3: GREEDY DECODING WITH HMM

In [67]:
```python
import json

# Loading HMM model
with open('hmm.json', 'r') as f:
    hmm_model = json.load(f)
transition_probs = hmm_model['transition']
emission_probs = hmm_model['emission']

# Function to predict the tag using greedy decoding
def predict_tag(word, prev_tag, transition_probs, emission_probs):
    # Initialize max probability and best tag variables
    max_prob = 0
    best_tag = None
    for tag in emission_probs:
        # Calculate the emission probability
        emission_prob = emission_probs[tag].get(word, 0)
        # Calculate the transition probability
        transition_prob = transition_probs.get(prev_tag, {}).get(tag, 0)
        # Calculate the combined probability
        prob = emission_prob * transition_prob
        if prob > max_prob:
            max_prob = prob
            best_tag = tag
    return best_tag if best_tag else 'NN'  # Default to 'NN' if no tag found

# Read development data and predict tags
output_lines = []
with open(r"C:\Users\adity\Desktop\NLP\CSCI544_HW3\hw3\data\dev", 'r') as file:
    prev_tag = '<s>'  # Start of sentence tag
    for line in file:
        if line.strip():
            parts = line.strip().split('\t')
            if len(parts) == 3:
                index, word, _ = parts  # Ignore the actual tag
            else:
                raise ValueError(f"Line does not contain three tab-separated values: {line}")
            # Predict the tag
            predicted_tag = predict_tag(word, prev_tag, transition_probs, emission_probs)
            output_lines.append(f"{index}\t{word}\t{predicted_tag}\n")
            prev_tag = predicted_tag
        else:
            output_lines.append("\n")
            prev_tag = '<s>'


# Writng predictions to a file
with open('greedy.out', 'w') as out_file:
    out_file.writelines(output_lines)
```

In [68]:
```python
# Printing first 10 lines of the greedy.out file to check the result
with open('greedy.out', 'r') as f:
    for _ in range(10):
        print(f.readline(), end='')
```

```
1       The     NN
2       Arizona NNP
3       Corporations    NNS
4       Commission      NNP
5       authorized      VBD
6       an      DT
7       11.5    CD
8       %       NN
9       rate    NN
10      increase        NN
```

In [69]:
```python
# Evaluating accuracy
!python "C:\Users\adity\Desktop\NLP\CSCI544_HW3\hw3\eval.py" -p "C:\Users\adity\greedy.out" -g "C:\Users\adity\Desktop\NLP\CSCI544_HW3\hw3\data\dev"
```

```
total: 131768, correct: 117669, accuracy: 89.30%
```

TASK 4: VITERBI DECODING WITH HMM

In [70]:
```python
import json
import numpy as np

# Load HMM model
with open('hmm.json', 'r') as f:
    hmm_model = json.load(f)
transition_probs = hmm_model['transition']
emission_probs = hmm_model['emission']
states = list(emission_probs.keys())  # Assuming all states emit at least one word

#  Viterbi algorithm
def viterbi(obs, states, trans_p, emit_p):
    V = [{}]
    path = {}

    # Initializing start probabilities with default if '<s>' not found
    start_p = {state: 1/len(states) for state in states} if '<s>' not in trans_p else trans_p['<s>']

    # Base case: Initialize probabilities for first observation
    for st in states:
        V[0][st] = start_p.get(st, 0) * emit_p[st].get(obs[0], 0)
        path[st] = [st]

    # Run Viterbi for t > 0
    for t in range(1, len(obs)):
        V.append({})
        newpath = {}

        for cur_state in states:
            # small probability for unseen transitions/emissions
            max_prob = max((V[t-1][prev_state] * trans_p[prev_state].get(cur_state, 1e-6) * emit_p[cur_state].get(obs[t], 1e-6), prev_state) for prev_state in states)
            V[t][cur_state], state = max_prob
            newpath[cur_state] = path[state] + [cur_state]

        path = newpath

    # Choosing ending state with highest probability
    n = len(obs) - 1
    # check if last column in V is empty for ValueError
    if all(V[n][state] == 0 for state in states):
        # If all probabilities are zero, we can't find a max

        prob, state = 0, states[0]
    else:
        prob, state = max((V[n][state], state) for state in states)

    return (prob, path[state])


# Read dev data, apply Viterbi algorithm, and writing predictions
output_lines = []
dev_file_path = r"C:\Users\adity\Desktop\NLP\CSCI544_HW3\hw3\data\dev"
with open(dev_file_path, 'r') as file:
    sentence = []
    for line in file:
        if line.strip():
            index, word, _ = line.strip().split('\t')
            sentence.append(word)
        else:
            if sentence:
                _, tags = viterbi(sentence, states, transition_probs, emission_probs)
                output_lines.extend([f"{i+1}\t{word}\t{tag}\n" for i, (word, tag) in enumerate(zip(sentence, tags))])
            output_lines.append("\n")
            sentence = []

# Writing predictions to the viterbi.out file
with open('viterbi.out', 'w') as out_file:
    out_file.writelines(output_lines)
```

In [71]:
```python
# Print first few lines of the viterbi.out file to verify the output
with open('viterbi.out', 'r') as f:
    lines = [next(f) for _ in range(10)]
    print(''.join(lines))
```

```
1       The     DT
2       Arizona NNP
3       Corporations    NNP
4       Commission      NNP
5       authorized      VBD
6       an      DT
7       11.5    CD
8       %       NN
9       rate    NN
10      increase        NN
```

In [72]:
```python
# Evaluating accuracy
!python "C:\Users\adity\Desktop\NLP\CSCI544_HW3\hw3\eval.py" -p "C:\Users\adity\viterbi.out" -g "C:\Users\adity\Desktop\NLP\CSCI544_HW3\hw3\data\dev"
```

```
'1\tThat\tDT' '38\t.\t.' 131751
total: 131751, correct: 116916, accuracy: 88.74%
```

In [ ]:

BRIEF EXPLANATIONS:

TASK 1: VOCABULARY CREATION

1. **Read Training Data**: Open the training data file and read it line by line, skipping any blank lines.
2. **Count Word Frequencies**: Use a dictionary to count each word's occurrence in the data.
3. **Handle Rare Words**: Create a special token **<unk>** for rare words that appear less than the threshold (three times).
4. **Sort Vocabulary**: Sort the vocabulary based on frequency in descending order.
5. **Index Vocabulary**: Assign an index to each word in the sorted vocabulary list.
6. **Write to File**: Write the indexed vocabulary to **vocab.txt**, with each line containing the index, word, and count, separated by tabs.
7. **Output Vocabulary Size**: Print the total number of unique words in the vocabulary (excluding **<unk>**).
8. **Output <unk> Occurrences**: Print the total number of occurrences that have been replaced by **<unk>**.

TASK 2: MODEL LEARNING

1. **Initialize Dictionaries**: Use **defaultdict(Counter)** to track transitions and emissions, and **Counter** for tag occurrences.
2. **Read Training Data**: Process the training data file line by line, updating transition and emission counts.
3. **Track Transitions**: For each tag pair, increment transition counts from the previous tag to the current tag.
4. **Handle Emissions**: Increment emission counts for each word-tag pair.
5. **Reset at Sentence End**: Reset the previous tag marker at the end of each sentence to handle sentence boundaries correctly.
6. **Calculate Probabilities**: Compute the transition and emission probabilities using the counts.
7. **Create HMM Model**: Assemble the transition and emission probabilities into a dictionary representing the HMM.
8. **Write Model to File**: Output the HMM model to **hmm.json** in JSON format.
9. **Output Parameters**: Print the number of transition and emission parameters to check the model's complexity.

TASK 3: GREEDY DECODING WITH HMM

1. **Load HMM Model**: Import the previously trained HMM model from **hmm.json**, including transition and emission probabilities.
2. **Greedy Decoding Function**: Define **predict_tag** to determine the most probable tag for a word based on the maximum combined probability of the emission probability of the word and the transition probability from the previous tag.
3. **Read Development Data**: Process the development data file, applying the **predict_tag** function to each word to predict its tag.
4. **Handle Sentence Boundaries**: Use a special start-of-sentence tag **<s>** to reset context at the end of each sentence.
5. **Predict Tags**: Use the greedy algorithm to predict the part-of-speech tags for each word, defaulting to 'NN' if no suitable tag is found.
6. **Write Predictions to File**: Store the predicted tags in **greedy.out**, formatted with the word's index, the word itself, and its predicted tag, separated by tabs.

TASK 4: VITERBI DECODING WITH HMM

1. **Load HMM Model**: Import transition and emission probabilities from **hmm.json**.
2. **Define States**: Create a list of possible states (POS tags) based on emission probabilities.
3. **Viterbi Algorithm**: Implement the algorithm, initializing probabilities with a default value if the start symbol **'<s>'** is not found, to handle unseen words or transitions.
4. **Iterate Over Observations**: Calculate the maximum probability path for each state at every step in the sequence.
5. **Handle Unseen Transitions**: Assign a small probability for unseen transitions or emissions to avoid zero probabilities.
6. **Track Paths**: Keep a record of the path that leads to the highest probability for each state.
7. **Select Best Path**: After processing all observations, determine the path with the highest probability as the sequence of tags.
8. **Read and Predict**: Process the development data, apply the Viterbi algorithm, and store the predicted sequences.
9. **Write to File**: Output the predicted sequences to **viterbi.out** with the format of index, word, and predicted tag per line.