Task 1:

1. What is the selected threshold for unknown words replacement?

Sol: We have selected threshold value = 1 in order to improve the accuracy

2. What is the total size of your vocabulary ?

Sol: Total size of vocabulary is : 23182

We have kept threshold as 1 so all values below this threshold are considered as <unk>.

Below is a snapshot from vocab.txt. We have total 45 tags in our dictionary from train data.

```
twinkledhanak@Twinkles-Air hw2 % python3 HW2_Greedy_Twinkle.py
Task 1 started...
Vocabulary size is:  23182
Vocabulary file is created. Task 1 complete.
```

vocab.txt × HW2_Viterbi

vocab.txt

```
1   <unk>   0   20011
2   ,   1   46476
3   the 2   39533
4   .   3   37452
5   of  4   22104
6   to  5   21305
```

3. What is the total occurrences of the special token '< unk >' after replacement?

Sol: Since we have threshold value = 1; after replacing all of words below threshold with <unk>, we get the following count: 20011 for <unk>

Task 2:

How many transition and emission parameters in your HMM?

Sol: We have 2070 records in transmission dictionary and 1063245 records in our emission dictionary.

```
twinkledhanak@Twinkles-Air hw2 % python3 HW2_Greedy_Twinkle.py
Task 1 started...
Vocabulary size is:  23182
Vocabulary file is created. Task 1 complete.
Task 2 started...
Size of Transition dict :  2070
Size of Emission dict:  1063245
File hmm.json with Transition and Emission probabilities created. Task 2 complete.
```

Task 3:
What is the accuracy of Greedy HMM decoding on the dev data?
Sol: With Greedy HMM, we are achieving 92.36 % accuracy on dev data.

```
twinkledhanak@Twinkles-Air hw2 % python3 HW2_Greedy_Twinkle.py
Task 1 started...
Vocabulary size is:  23182
Vocabulary file is created. Task 1 complete.
Task 2 started...
Size of Transition dict :  2070
Size of Emission dict:  1063245
File hmm.json with Transition and Emission probabilities created. Task 2 complete.
Task 3 started...
For Dev data, Correct predicted tags:  121705  ,Total no. of tags:  131768  ,Greedy HMM Accuracy:  92.36309270839658
Now Predicting parts-of-speech for test data using Greedy HMM.
File greedy.out created. Task 3 complete.
```

Output your predictions on test data in a file greedy.out.
Sol: We have created a file greedy.out and dumped all predictions data into that file for Greedy hmm.

greedy.out ×

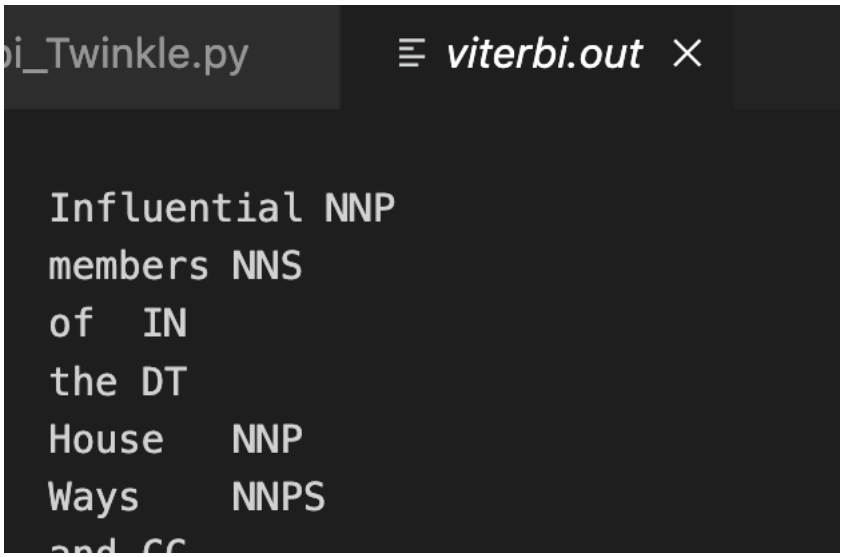
greedy.out

```
1   1   Influential NNP
2   2   members NNS
3   3   of  IN
4   4   the DT
5   5   House   DT
6   6   Ways    DT
```

Task 4:
What is the accuracy of Viterbi algorithm on the dev data?
Sol: With Viterbi algorithm, we are achieving 93.74% accuracy on our dev data.

```
twinkledhanak@Twinkles-Air hw2 % python3 HW2_Viterbi_Twinkle.py
Task 4.1 started...
For Dev data, Correct predicted tags:  123528  Total no. of tags:  131768  ,Viterbi Accuracy::  93.7465849068059
Now Predicting parts-of-speech for test data using Viteri.
File viterbi.out created. Task 4.2 complete.
twinkledhanak@Twinkles-Air hw2 %
```

Output your predictions on test data in a file viterbi.out .
Sol: We have created a file Viterbi.out and dumped all predictions data into that file for Greedy hmm.



Approach for good accuracy:

1. For Greedy, implementing the algorithm correctly improved the accuracy. We have to consider the right number of states transitions and their emission probabilities to get good accuracy.
2. For Viterbi algorithm, we implemented the basic algorithm and that helped achieved good accuracy (~ 87% initially). We have two areas of improvement implemented -

For viterbi, if a given word is unknown, not present in train data corpus, viterbi algorithm tags the pos with the first one available in its dictionary (postag_dict in code). SO to improve on probabilities for such unknown word, we have implemented Laplace smoothing at one place for emission probability.
Second thing is we have implemented a custom function to predict what would be that POS tag for a word that is not present in train data. We have compiled different regex strategies for nouns, verbs and adjectives and tried to assign a correct tag to a unknown word in order to improve accuracy.

```
# An attempt at Laplace smoothing for Emission values so we better estimate the probabilities of unknown tags
emission[emission_key] = (obs_state_dict[state_key]+1) /(postag_dict[i] + len(postag_dict) )
#print("Dictionary: ",emission_key," ,value: ",emission[emission_key],"state key: ",state_key," State dict value: ",ob
```

```
gerund = re.compile(r'.*ing$')
past_tense_verbs = re.compile(r'.*ed$')
singular_present_verbs = re.compile(r'.*es$')
modal_verbs = re.compile(r'.*ould$')
possessive_nouns = re.compile(r'.*\'s$')
plural_nouns = re.compile(r'.*s$')
cardinal_numbers = re.compile(r'^-?[0-9]+(.[0-9]+)?$')
articles_determinants = re.compile(r'(The|the|A|a|An|an)$')
adjectives = re.compile(r'.*able$')
nouns_formed_from_adjectives = re.compile(r'.*ness$')
adverbs = re.compile(r'.*ly$')
nouns = re.compile(r'.*')
```