EX.NO.: 22
DATE: 28.03.2025

VITERBI ALGORITHM FOR POS TAGGING

To implement the Viterbi Algorithm for POS tagging and determine the most probable sequence of tags for a given sentence.

PROCEDURE:

1. Define the states (POS tags) and the input sentence.
2. Initialize transition probabilities, emission probabilities, and initial probabilities.
3. Use the Viterbi algorithm to compute the most probable sequence of POS tags:
    a. Initialize the first word's probabilities using the initial probabilities.
    b. Recursively compute the probability of each POS tag at each step, considering previous states.
    c. Backtrack to find the best sequence.
4. Print the best POS tag sequence.

CODE AND OUTPUT

```
import numpy as np

# Define states (POS tags)
states = ['NNP', 'MD', 'VB', 'DT', 'NN']  # Proper Noun, Modal, Verb, Determiner, Noun

# Transition Probability Matrix (A)
transition_prob = {
    'NNP': {'NNP': 0.0, 'MD': 0.5, 'VB': 0.0, 'DT': 0.0, 'NN': 0.5},
    'MD': {'NNP': 0.0, 'MD': 0.0, 'VB': 1.0, 'DT': 0.0, 'NN': 0.0},
    'VB': {'NNP': 0.0, 'MD': 0.0, 'VB': 0.0, 'DT': 0.5, 'NN': 0.5},
    'DT': {'NNP': 0.0, 'MD': 0.0, 'VB': 0.0, 'DT': 0.0, 'NN': 1.0},
    'NN': {'NNP': 0.0, 'MD': 0.0, 'VB': 0.0, 'DT': 0.0, 'NN': 0.0},
}

# Emission Probability Matrix (B)
emission_prob = {
    'NNP': {'Janet': 1.0, 'will': 0.0, 'back': 0.0, 'the': 0.0, 'bill': 0.0},
    'MD': {'Janet': 0.0, 'will': 1.0, 'back': 0.0, 'the': 0.0, 'bill': 0.0},
    'VB': {'Janet': 0.0, 'will': 0.0, 'back': 1.0, 'the': 0.0, 'bill': 0.0},
    'DT': {'Janet': 0.0, 'will': 0.0, 'back': 0.0, 'the': 1.0, 'bill': 0.0},
    'NN': {'Janet': 0.0, 'will': 0.0, 'back': 0.5, 'the': 0.0, 'bill': 0.5},
}

# Initial Probability Distribution
initial_prob = {'NNP': 1.0, 'MD': 0.0, 'VB': 0.0, 'DT': 0.0, 'NN': 0.0}

# Sentence to tag
sentence = ['Janet', 'will', 'back', 'the', 'bill']

# Viterbi Algorithm
V = [{}]  # Stores probability of best path to each state
path = {}
```

```python
# Initialize base case
for state in states:
    V[0][state] = initial_prob[state] * emission_prob[state].get(sentence[0], 0)
    path[state] = [state]
print(f"Step 0: {sentence[0]} {V[0]}")

# Run Viterbi for t > 0
for t in range(1, len(sentence)):
    V.append({})
    new_path = {}
    print(f"\nStep {t}: {sentence[t]}")

    for curr_state in states:
        (prob, prev_state) = max(
            (V[t - 1][prev_state] * transition_prob[prev_state].get(curr_state, 0) *
             emission_prob[curr_state].get(sentence[t], 0), prev_state)
            for prev_state in states
        )
        V[t][curr_state] = prob
        new_path[curr_state] = path[prev_state] + [curr_state]
        print(f"{curr_state}: max_prob={prob} from {prev_state}")

    path = new_path

# Find the best final state
final_prob, final_state = max((V[-1][state], state) for state in states)

# Get the best sequence
best_sequence = path[final_state]

print("\nBest POS Tag Sequence:")
for word, tag in zip(sentence, best_sequence):
    print(f"{word}: {tag}")
print("Final Probabilities:", V[-1])
```

```
NN: max_prob=0.0 from VB

Step 2: back
NNP: max_prob=0.0 from VB
MD: max_prob=0.0 from VB
VB: max_prob=0.5 from MD
DT: max_prob=0.0 from VB
NN: max_prob=0.0 from VB

Step 3: the
NNP: max_prob=0.0 from VB
MD: max_prob=0.0 from VB
VB: max_prob=0.0 from VB
DT: max_prob=0.25 from VB
NN: max_prob=0.0 from VB

Step 4: bill
NNP: max_prob=0.0 from VB
MD: max_prob=0.0 from VB
VB: max_prob=0.0 from VB
DT: max_prob=0.0 from VB
NN: max_prob=0.125 from DT

Best POS Tag Sequence:
Janet: NNP
will: MD
back: VB
the: DT
bill: NN
Final Probabilities: {'NNP': 0.0, 'MD': 0.0, 'VB': 0.0, 'DT': 0.0, 'NN': 0.125}
```

INFERENCE

The Viterbi Algorithm successfully finds the most probable POS tag sequence for a given sentence by maximizing the probability of transitions and emissions. The computed sequence aligns with linguistic expectations, demonstrating the effectiveness of HMM-based POS tagging.