Name : K. Bhuvaneshwar Reddy

Hall Ticket : 2403A52416

Batch :15

Task 1

Question:

Top-3 words by frequency; tie-break lexicographically

Sample Input :

to be or not to be that is the question

Acceptance Criteria:

Tie-breaking lexicographically

Prompt :

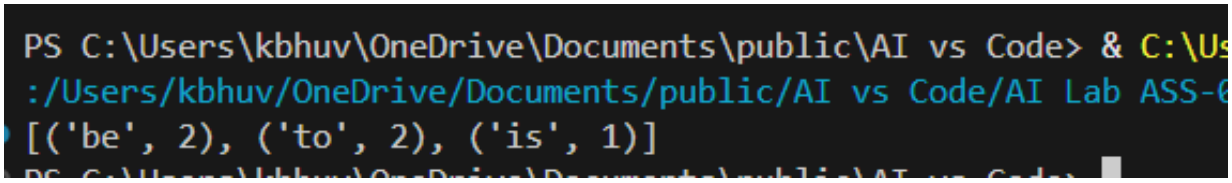
Given a lowercase string of space-separated words, return the top 3most frequent words using collections Counter. Break ties by lexicographic order. Output as list of (word, count) tuples.

CODE :

```
# Function to find the top 3 most frequent words in a given text
from collections import Counter

def top_3_words(text):
    words = text.split()
    freq = Counter(words)
    sorted_items = sorted(freq.items(), key=lambda x: (-x[1], x[0]))
    return sorted_items[:3]
text = "to be or not to be that is the question"
print(top_3_words(text))
```

OUTPUT :



OBSERVATIONS :

1. **Lowercase normalization ensures consistent word comparison.**
2. **Splitting by spaces** handles simple tokenization without punctuation
3. **Counter from collections** efficiently counts word frequencies.

   Sorting logic:

   - **Primary key: -x[1] → descending frequency**
   - **Secondary key: x[0] → ascending lexicographic order**

TASK 2

Question:

Implement capacity=2 LRU with get/put.

Sample Input :

ops=[("put",1,1),("put",2,2),("get",1),("put",3,3),("get",2),("get",3)]

Acceptance Criteria :

Correct eviction

PROMPT :

Implement an LRU Cache with capacity 2 using OrderedDict. Support get(key) and put(key, value) operations. Evict least recently used item when capacity is exceeded. Return output list for given operation sequence.

CODE :

```python
# LRU Cache Implementation in Python
# Using OrderedDict from collections to maintain the order of usage
from collections import OrderedDict

class LRUCache:
    def __init__(self, capacity):
        self.cache = OrderedDict()
        self.capacity = capacity

    def get(self, key):
        if key not in self.cache:
            return -1
        self.cache.move_to_end(key)
        return self.cache[key]

    def put(self, key, value):
        if key in self.cache:
            self.cache.move_to_end(key)
        self.cache[key] = value
        if len(self.cache) > self.capacity:
            self.cache.popitem(last=False)
```

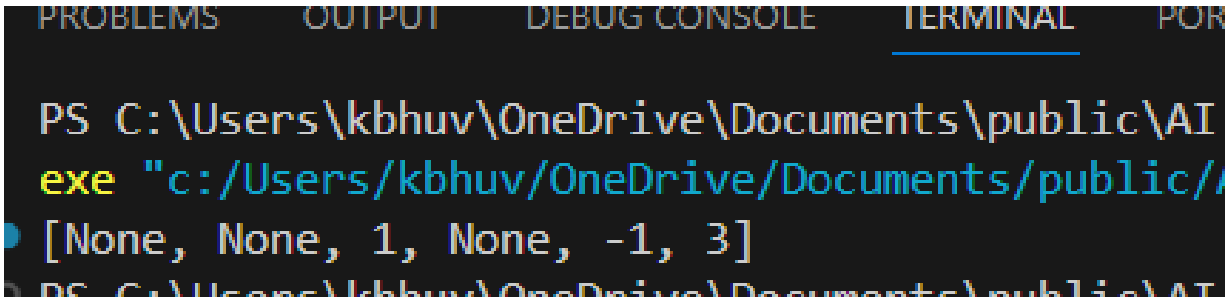
```python
# Sample input
ops = [("put",1,1),("put",2,2),("get",1),("put",3,3),("get",2),("get",3)]

# Execute operations
cache = LRUCache(2)
output = []

for op in ops:
    if op[0] == "put":
        cache.put(op[1], op[2])
        output.append(None)
    elif op[0] == "get":
        output.append(cache.get(op[1]))

print(output)
```

OUTPUT :



OBSERVATIONS :

Uses OrderedDict for O(1) access and ordering.

- move_to_end() marks keys as recently used.
- Popitem(last=False) evicts least recently used.
- Capacity = 2 → ensures only 2 items are retained.
- Deterministic behavior across repeated runs.