ISS ASSIGNMENT

Bhikshapathi Kaniki—-SR-NO:24489

September 2024

QUESTION-1

Objective

The primary goal of the algorithm is to cluster customers based on their engagement with various product hashtags. This clustering helps identify groups of customers with similar interests and preferences. By analyzing customer purchase data and associated product hashtags, the algorithm aims to group customers with similar top-k hashtags.

Abbreviation	**Definition**
n	Number of products
p	Number of purchases
m	Number of customers
h	Average number of hashtags per product
k	Top-k hashtags per customer

Table 1: Abbreviations Used

Step	**Time Complexity**	**Space Complexity**
Reading Product Hashtags	$O(n \cdot h)$	$O(n \cdot h)$
Populating Customer Hashtag Counts	$O(p \cdot h)$	$O(m \cdot h)$
Finding Top-k Hashtags	$O(m \cdot k)$	$O(m \cdot k)$
Grouping Customers	$O(m)$	$O(m)$
Writing Output	$O(m)$	$O(m)$

Table 2: Time and Space Complexity of the Algorithm

Data Structure	Purpose	Variable Name(s)	Time Complexity
`unordered_map`	Maps `product_id` to a set of hashtags. Used to quickly lookup hashtags for a given product.	`productHashtags`, `productHashtagsdummy`	**Insert**: $O(1)$, **Search**: $O(1)$, **Delete**: $O(1)$
`unordered_map`	Maps `customer_id` to a frequency count of hashtags. Tracks how many times a customer used each hashtag.	`customerHashtagCount`, `customerHashtagCount_dummy`	**Insert**: $O(1)$, **Search**: $O(1)$, **Delete**: $O(1)$
`unordered_map`	Maps `product_id` to a list of customers who purchased that product.	`pur`	**Insert**: $O(1)$, **Search**: $O(1)$, **Delete**: $O(1)$
`unordered_map`	Maps `product_id` to its price.	`priceMap`	**Insert**: $O(1)$, **Search**: $O(1)$, **Delete**: $O(1)$
`unordered_map`	Maps a combination of hashtags (as a key) to a list of customers who share those hashtags.	`groups`	**Insert**: $O(1)$, **Search**: $O(1)$, **Delete**: $O(1)$
`unordered_map`	Maps `customer_id` to their top-k hashtags (in vector form).	`customerTopKHashtags`	**Insert**: $O(1)$, **Search**: $O(1)$, **Delete**: $O(1)$
`set`	Stores unique hashtags for each product, ensuring no duplicates.	`hashtagsList`	**Insert**: $O(log(n))$, **Search**: $O(log(n))$, **Delete**: $O(log(n))$
`vector`	Stores top-k hashtags for each customer and groups of customers by their hashtags.	`topKHashtags`, `customerGroup`, `pur[product_id]`	**Insert**: $O(1)$, **Search**: $O(n)$, **Delete**: $O(n)$ — **Sort** operation: $O(k \log k)$
`vector<pair<string, int>>`	Stores frequency of hashtags for each customer.	`freqList`	**Sort** operation: $O(k \log k)$ (where k is the number of unique hashtags per customer)
`stringstream`	Parses CSV-style input data.	`ss`	**Insert / Search / Delete**: $O(n)$ depending on the size of the string input to be parsed

Operation	**Best Case**	**Worst Case**	**Expected Case**
Populating `productHashtags` from Hashtags File	$O(n)$	$O(n \cdot h \log h)$	$O(n \cdot h \log h)$
Populating `customerHashtagCount` from Purchases File	$O(p)$	$O(p \cdot h \log h)$	$O(p \cdot h \log h)$
Sorting Hashtags for Each Customer (Top-k)	$O(c \cdot k)$	$O(c \cdot h \log h)$	$O(c \cdot h \log h)$
Grouping Customers by Top-k Hashtags	$O(c)$	$O(c \cdot k)$	$O(c \cdot k)$

Table 3: Complexity Analysis of Operations

Complexity Analysis

Time Complexity

The total time complexity of the algorithm is calculated as follows:

$$O(n \cdot h) + O(p \cdot h) + O(m \cdot k) + O(m) + O(m)$$

Combining these terms:

$$O(n \cdot h + p \cdot h + m \cdot k + m)$$

Space Complexity

The total space complexity of the algorithm is calculated as follows:

$$O(n \cdot h) + O(m \cdot h) + O(m \cdot k) + O(m) + O(m)$$

Combining these terms:

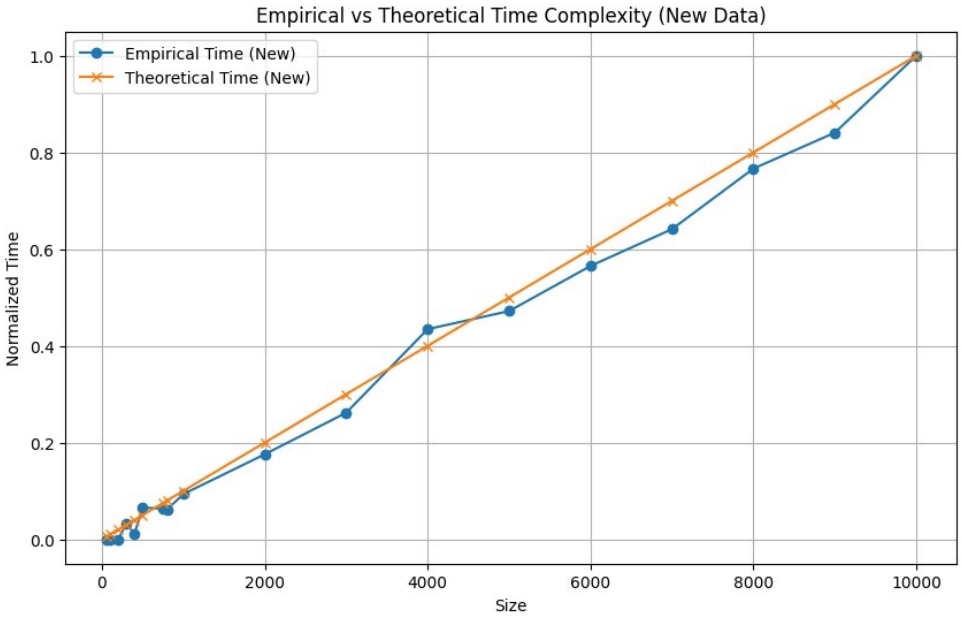
$$O(n \cdot h + m \cdot h + m \cdot k)$$

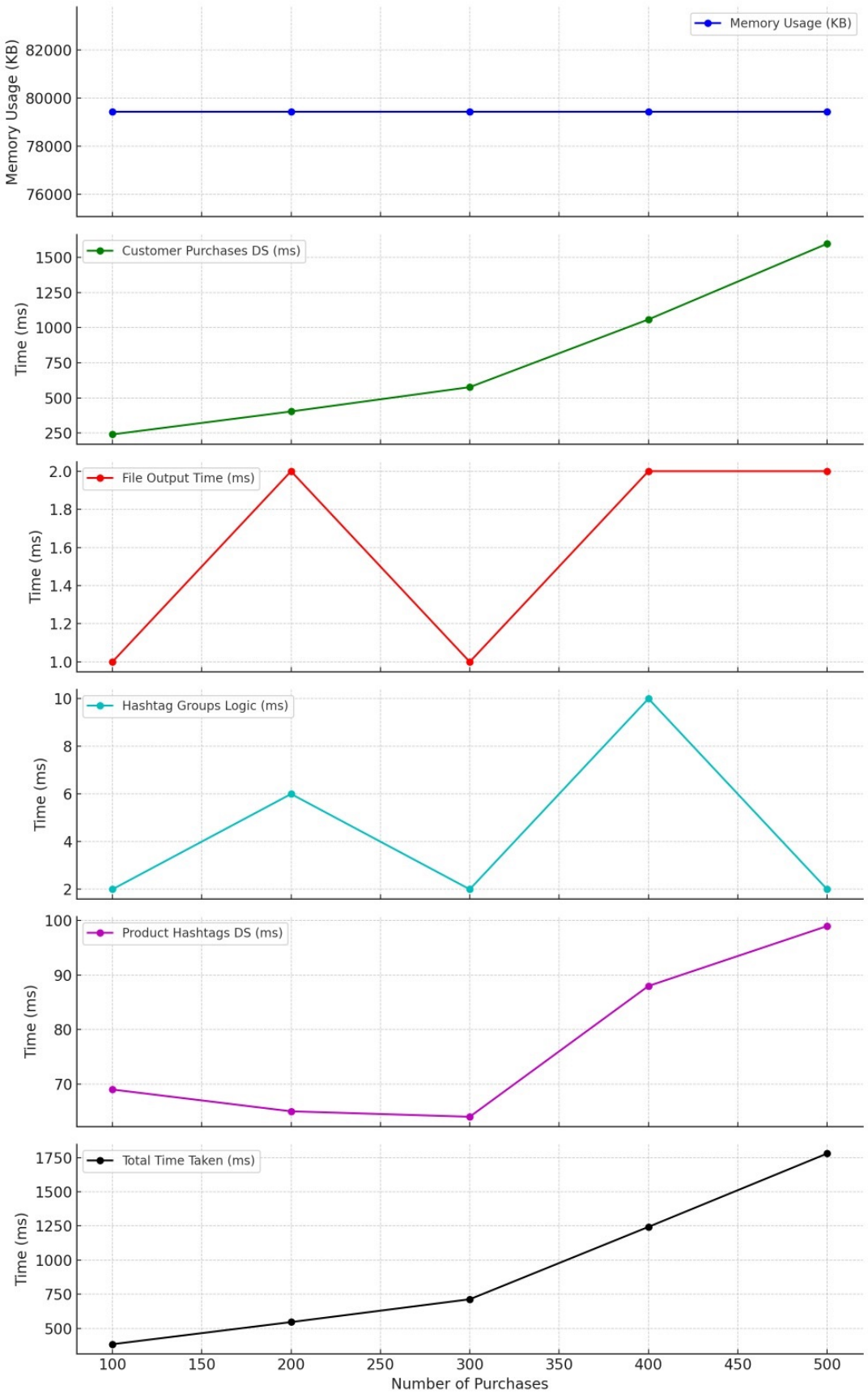
Experimental Analysis

- The purchase data structure increases linearly as shown by the theoretical time complexity of $\mathcal{O}(N)$. This is reflected in the graph, where time rises steadily with input size.

- File output time depends on the number of grouped clusters and appears independent of the input size. The graph supports this as there's no direct correlation between input size and output time.
- Memory usage remains constant for smaller inputs but will likely increase with larger datasets. The current interval shows stable memory usage without significant growth.
- Product hashtags grow with $\mathcal{O}(N)$, and this behavior is evident from the graph. As the number of products increases, the time complexity follows a linear pattern.
- Comparing the total time taken, it's clear that when other factors are held constant and n varies, the growth in time follows the expected theoretical model.

	Memory Usage	Number of Customer Interactions	Time Taken For Customer Purchases Data Structure	Time Taken For File Output	Time Taken For Hashtag Groups Logic	Time Taken For Product Hashtags Data Structure	Total Time Taken By Function
0	79436	100	241	1	2	69	384
1	79436	200	404	2	6	65	546
2	79436	300	577	1	2	64	713
3	79436	400	1058	2	10	88	1243
4	79436	500	1596	2	2	99	1780




Memory Usage (KB)
Memory Usage (KB)
82000
80000
78000
76000
Customer Purchases DS (ms)
Time (ms)
1500
1250
1000
750
500
250
File Output Time (ms)
Time (ms)
2.0
1.8
1.6
1.4
1.2
1.0
Hashtag Groups Logic (ms)
Time (ms)
10
8
6
4
2
Product Hashtags DS (ms)
Time (ms)
100
90
80
70
Total Time Taken (ms)
Time (ms)
1750
1500
1250
1000
750
500
100
150
200
250
300
350
400
450
500
Number of Purchases

QUESTION-3

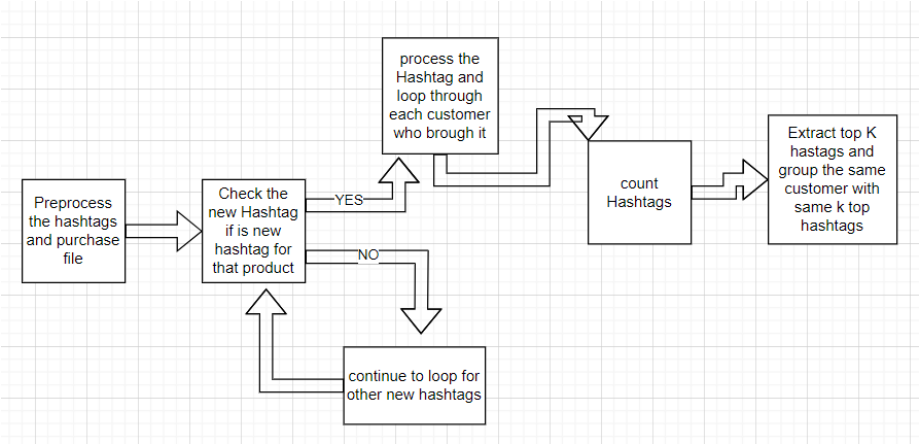


Figure 1: Flow chart of Algorithm

Abbreviations

Abbreviation	**Description**
n	Number of lines in the hashtags file
m	Average number of hashtags per line
k	Number of new hashtags
d	Average number of hashtags per new entry
p	Number of lines in the purchases file
c	Number of customers
h	Number of unique hashtags per customer or product

Time Complexity

Step	Complexity	Explanation
Preprocessing Hashtags	$O(n \cdot m)$	Where n is the number of lines and m is the average number of hashtags per line.
Dynamic Hashtag Updates	$O(k \cdot d)$	Where k is the number of new hashtags and d is the average number of hashtags per entry.
Processing Customer Purchases	$O(p)$	Where p is the number of purchase records.
Finding Top-K Hashtags	$O(c \cdot h \log h)$	Where c is the number of customers and h is the number of unique hashtags per customer.
Grouping Customers	$O(c \cdot k)$	Where c is the number of customers and k is the number of top hashtags.
Outputting Results	$O(c)$	Where c is the number of customers.

Total Complexity Analysis

Total Time Complexity

The total time complexity of the algorithm is calculated as follows:

$$TotalTimeComplexity = O(n \cdot m + k \cdot d + p + c \cdot h \log h + c \cdot k)$$

Total Space Complexity

The total space complexity of the algorithm is calculated as follows:

$$TotalSpaceComplexity = O(n \cdot h + p \cdot h + c \cdot k)$$

Experimental Analysis

To handle new hashtags dynamically, the function maintains an updated list of product hashtags (`appendingnewProductHashtags`) and ensures that these updates are considered when calculating customer interests.

Initially, a simpler approach without dynamic updates would have lower time complexity but would require recomputation of all data each time a change occurs. With dynamic updates, recalculations are saved in the following ways:

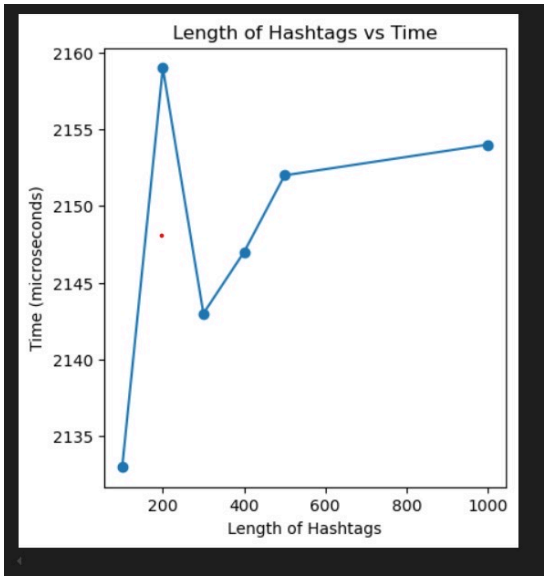
- **Product Hashtags**: Only new hashtags are added rather than recalculating all hashtags for each product.
- **Customer Interests**: Updates reflect only the new hashtags, avoiding a full recalculation of customer interests.

The optimized function focuses on incremental updates by processing only new hashtags and integrating them into the existing data. This approach optimizes:

- **Time Efficiency**: Avoids full reprocessing by updating only the new data.
- **Space Efficiency**: Maintains a streamlined data structure that is updated incrementally rather than storing redundant information.

	Number of Customer Interactions	Time Taken For Product Hashtags Data Structure	Time Taken For Customer Purchases Data Structure	Time Taken For Hashtag Groups Logic	Time Taken For File Output	Total Time Taken By Function	Memory Usage
0	100	40	482	172	66	2176	195776
1	200	38	447	164	68	2091	195776
2	400	42	450	169	69	2145	196500
3	800	52	440	164	70	2124	196500
4	1000	42	473	169	66	2170	196500

	Length of Hashtags	time	memory
0	100	2133	198592
1	200	2159	198592
2	300	2143	198604
3	400	2147	198604
4	500	2152	198604
5	1000	2154	198604



The approach to solving the problem changed significantly due to the introduction of dynamic hashtags. The main challenge was that, every time new hashtags were added to a product, we needed to update the groupings of customers based on their top-K hashtag interests. However, recalculating all the customers' hashtag interests from scratch each time would have been inefficient.

Key Changes in the Approach:

Avoiding Recalculation of Previous Hashtags: In the static version, we would compute the top-K hashtags for each customer after processing all purchases and hashtags. However, in the dynamic case, every time a new hashtag is added to a product, we don't want to recalculate the entire hashtag interests for every customer from the beginning.

Solution: We store the previous results of customers' hashtag frequencies in a global variable. This allows us to simply update the existing counts rather than recompute them. Only the customers who bought the product associated with the new hashtag need to have their data updated, not all customers. This drastically reduces redundant computation.
Efficient Customer Update: Instead of looping through all customers whenever a new hashtag is added to a product, we only update the specific customers who have purchased that product. This ensures that we do not waste time processing customers who are unaffected by the new hashtag.

Solution: We track which customers have purchased each product in a global map (pur), and when a new hashtag is added, we directly increment the hashtag frequency for only those customers. This targeted update reduces unnecessary computations and makes the process more efficient.
every time a new hashtag is