Programming Assignment 1

K Most Popular Words

Justin Li & Kyle Mondina

Introduction

The objective of this analysis is to design and implement an efficient code to determine the top K most frequent words in a given dataset, while also analyzing the impact of input size and algorithm efficiency on performance. The analysis will be conducted on three different datasets of varying sizes, ranging from 300MB to 16GB. In each of these datasets, we applied two approaches in finding the K most frequent words. For each approach we measured the runtime, cpu utilization, and memory utilization of the program. The results of the analysis will provide insights into the factors that affect the performance of the code. The code used to solve this problem is written in Python.

System Specification

CPU Cores - 4
Logical Processors - 8
RAM - 16GB

Methodology

In order to solve the problem of determining the top K most frequent words in a given dataset, we used two different approaches - dictionary-based and heap-based solution.

For preprocessing, we extracted the words from each dataset and removed the provided stopwords. As the size of the datasets is quite large, we used a generator instead of a list to store the extracted words to avoid running out of memory.

Approach 1: Dictionary based solution

In this approach, we used a dictionary to keep track of the frequency of each word in the dataset. For each word in the dataset, we checked if it was already present in the dictionary. If it was, we incremented its frequency by 1, otherwise, we added it to the dictionary with a frequency of 1. Finally, we sorted the dictionary by frequency in descending order and returned the top K words.

The dictionary-based solution has a time complexity of O(n) for preprocessing and O(n log n) for sorting, where n is the number of words in the dataset. The advantage of using a dictionary is that it provides a very fast look-up time for words, making it an ideal data structure for this problem. Furthermore, the dictionary can be sorted easily to get the top K words.

Approach 2: Counter and Heap based solution

In this approach, we use a Counter structure instead of a dictionary and a heap based solution instead of sorting the dictionary to find the top k most frequent words. We first use a counter to count the frequency of each word in our generator. . A counter is a dictionary subclass that provides a convenient way to count occurrences of elements in a sequence. We then use a heap to find the k most popular words.

A heap is a binary tree-based data structure where each node has a value that is greater than or equal to (in a max heap) or less than or equal to (in a min heap) its child nodes. In a max heap, the root node has the highest value in the heap, and in a min heap, the root node has the lowest value in the heap.

In this approach, we create a max heap where each node represents a word and its count in the sequence. Initially, the heap is empty, and we add the first k words from the sequence to the heap along with their corresponding counts. We then perform a heapify operation on the heap, which reorders the nodes in the heap such that the root node has the highest count and all the child nodes have lower counts.

Next, we iterate over the remaining words in the sequence and check if their counts are higher than the count of the root node in the heap. If the count is higher, we remove the root node from the heap and add the new word and its count to the heap. We then perform another heapify operation on the heap to reorder the nodes.

This process continues until we have processed all the words in the sequence. At the end, the heap contains the k most common words in the sequence, with the root node being the word with the highest count. We can then extract the k most common words from the heap by repeatedly extracting the root node and adding it to a list until we have extracted k nodes.

The heap-based solution has a time complexity of O(n log k), where n is the number of words in the dataset and k is the value of K.

Results and Analysis

300MB Dataset

Dictionary Solution	Heap Solution

Runtime	90.726990 seconds
CPU Utilization	99.90%
Memory Usage	8.89 MB

Word	**Frequence Count**
European	318733
Mr	210680
would	180717
also	178517
Commission	172277
must	156781
President	150884
Union	125350
Member	121252
Parliament	119643

Runtime	86.996652 seconds
CPU Utilization	99.90%
Memory Usage	2.64 MB

Word	**Frequence Count**
European	318733
Mr	210680
would	180717
also	178517
Commission	172277
must	156781
President	150884
Union	125350
Member	121252
Parliament	119643

2.5GB Dataset

Dictionary Solution

Runtime	791.674194 seconds
CPU Utilization	99.90%
Memory Usage	9.47 MB

Word	**Frequence Count**
said	2612121
would	910779
one	812228

Heap Solution

Runtime	756.754958 seconds
CPU Utilization	100.00%
Memory Usage	3.95 MB

Word	**Frequence Count**
said	2612121
would	910779
one	812228

also	708992
Mr	658579
people	643920
year	637848
de	622408
years	597943
two	584332

also	708992
Mr	658579
people	643920
year	637848
de	622408
years	597943
two	584332

16GB Dataset

Dictionary Solution

Runtime	2044.504627 seconds
CPU Utilization	99.90%
Memory Usage	39.83 MB

Word	**Frequence Count**
said	9494922
would	3215196
one	2734522
also	2507452
people	2221022
year	2133758
two	2000477
Mr.	1995816
first	1972701
years	1926570

Heap Solution

Runtime	2197.506550 seconds
CPU Utilization	99.90%
Memory Usage	31.57 MB

Word	**Frequence Count**
said	9494922
would	3215196
one	2734522
also	2507452
people	2221022
year	2133758
two	2000477
Mr.	1995816
first	1972701
years	1926570

Analysis

Based on the results of the experiments, it can be observed that both the dictionary-based and heap-based solutions were able to extract the top K most frequent words from the datasets. However, the performance of these algorithms varied based on the size of the dataset and the value of K.

For the smaller dataset of 300MB, the heap-based solution outperformed the dictionary-based solution in terms of runtime and memory usage. As the size of the dataset increased to 2.5GB and 16GB, the heap-based solution continued to perform well in terms of runtime and memory usage.

The difference in performance can be attributed to the difference in the algorithm and data structures used in the two approaches. The dictionary-based solution creates a dictionary where the keys are words and the values are their corresponding frequencies. This requires us to store the entire dictionary in memory, which can take up a significant amount of memory for large datasets. Specifically, the memory required to store the dictionary is proportional to the number of unique words in the dataset.

On the other hand, the heap-based solution only needs to store the top k words and their frequencies in the heap. This is because the heap-based solution uses a heap to keep track of the k most common words. The heap only needs to store k elements at any given time, which is much smaller than the total number of unique words in the dataset. As a result, the heap-based solution is more memory-efficient for large datasets.

Conclusion

The objective of the analysis was to design and implement an efficient code to determine the top K most frequent words in a given dataset, while also analyzing the impact of input size and algorithm efficiency on performance. Two different approaches were used to solve this problem - dictionary-based and heap-based. For each approach, the runtime, CPU utilization, and memory utilization of the program were measured on three different datasets of varying sizes ranging from 300MB to 16GB. The dictionary-based solution had a time complexity of O(n log n) for sorting, while the heap-based solution had a time complexity of O(n log k), where n is the number of words in the dataset, and k is the value of K. The heap-based solution was found to be more efficient than the dictionary-based solution, particularly on larger datasets. Overall, the results of the analyses provide insights into the factors that affect the performance of the code, which can be used to optimize the code further for similar tasks.