SDS Assignment2

IMT2020548 Tejas Sharma

September 2023

1 Question 1

Problem statement - Create a data generator that generates an event consisting of timestamp, integer and a character. The input to the generator is the desired throughput and the duration of runtime. Also create a consumer process which searches for a palindrome occurrence in the stream.

1.1 Generator logic

Use of the random choice function in python to generate events. The timestamp is assigned from the system time. Logic -
While we still have to generate data, use the random and system time functions to generate tuple and write to a csv file. The moment a second has occurred, start writing to a new file.
Initially, the maximum possible throughput achievable through this method was 25 tuples per second into a csv file.
Improvements -
First generate all tuples corresponding to the given throughput input and store it in a list and then insert the list into the csv file.
This led to an improvement in the throughput from 25 to 10000 tuples inserted per second.
On a side note, to ensure that there are some palindromes found in the result, I have added a probability value as threshold. Based on this value, we either add a random letter to the stream or add a letter present in the palindrome to ensure that some palindromes are found.

1.2 Consumer logic

Here the task is to count the number of occurrences of a simple palindrome with some constraints in the occurrence of the letters in it.
The idea behind the algorithm -
Given the stream data, first concatenate all characters in it so that we can search for the palindrome in it.

Suppose the palindrome to be searched is - "ABBA" with the following constraints - A[¡=2] and B[¡=1]. First, for each pair of matching letters in the palindrome, find the index of all possible occurrences in the concatenated string. Suppose in the string, the A-A pair occurs in - [(1, 10), (1, 12), (10, 16), (12, 16)] and B-B pair occurs in [(4,5), (4, 11), (5, 11)].
Now since B-B is always inside A-A in the palindrome, the intervals in B-B pair list should be inside some interval in the A-A pair list for it to be a valid palindrome. So, we have to start with the innermost letter and proceed outwards in the palindrome and eliminate all those occurrences where the inner intervals are not completely inside the outer interval. Now, we have to search recursively to find all possible palindromes by eliminating possibilities.

```
def findPairs(pattern, s, bounds, df):
    n = len(pattern)
    # print(s)
    m = len(s)
    pairs = {}
    for i in range(len(bounds)):
        for j in range(len(s)):
            if(s[j] == bounds[i][0]):
                k = j+1
                while(k<len(s) and df.loc[k,'timestamp'] <= bounds[i][1]+df.loc[j, 'timestamp']):
                    if(s[k] == bounds[i][0]):
                        if i not in pairs:
                            pairs[i] = []
                            pairs[i].append((j,k))
                        else:
                            pairs[i].append((j, k))
                    k+=1
                # print("found all in the range of this")
    # print(pairs)
    return pairs
```

The above picture is the logic to calculate all pairs.

```
def findCount(curr, prev, k):
    print("Intial Searching between: {} and {}".format(curr, prev))
    if(prev == []):
        return len(curr)
    temp = copy.deepcopy(prev)
    ans = 0
    for i in curr:
        print("Searching between: {} and {}".format(i, prev))
        for j in prev:
            if(i[0]>=j[0] and i[1]<=j[1]):
                pass
            else:
                temp.remove(j)
        if(k-1 < 0):
            ans += findCount(temp, [], k-1)
        else:
            ans += findCount(temp, pairs(k-1), k-1)
        temp = copy.deepcopy(prev)
        print('Temp is: {}'.format(temp))
        print('At the end of this iteration the partial answer is: {}'.format(ans))
    return ans
```

The above picture is to count the distinct palindrome pairs from this.

1.3 Driver logic

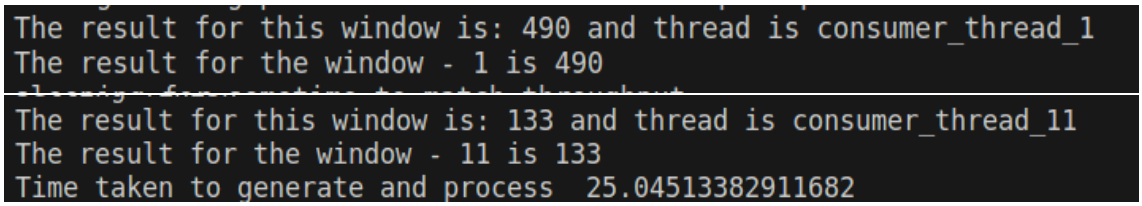
1.3.1 Method 1

In this method, we wait for all data to come and then release results for every 10 second window. We first call the generator thread and wait for it to complete and then call the consumer process and release results accordingly. For complexity reasons, I have set the generator probability threshold as 0.05 and the throughput as 20000 tuples. The pattern we're looking for is - A[¡=2]B[¡=1] and I run it for 20 seconds.
The result for it is -

```
The 1th window result is: 13623
The 2th window result is: 7583
Consumer process has computed on the whole dataset in...105.9133529663086 seconds
21206
Time taken to generate and process:  125.93009877204895
```

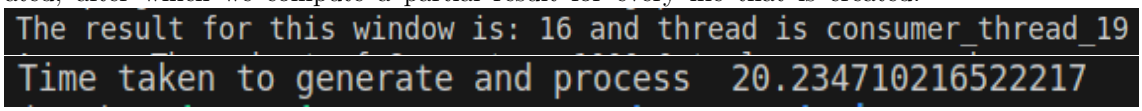
1.3.2 Method 2

In this method, we call the consumer function after every 10 files are created. This synchronization is maintained by keeping a threading event in the generator that is set true every 10 files created. This event is listened at the driver and when true, it calls the consumer process.



1.3.3 Method 3

In this method, we first check what is the maximum size among all constraints in the palindrome. Then we compute our partial results for a window of size equal to that maximum size. So we first wait for those many files to be created, after which we compute a partial result for every file that is created.



2 Question 2

Here, we just need to experiment with blooms filter. First, we create a normal function which takes in the streaming words data and without using an in memory data structure, we perform a lookup every time and add a unique word to a new file. This process is time consuming and for the entire file, it takes 90 minutes on the system to run.
This efficiency can be improved by using a blooms filter. I use the mmh3 hash function as the independent hash functions required for this filter. Here, we experiment with different false positivity rate as we already know the expected number of elements to be inserted into the filter.
As we lower the FPR, we observe that the bitmap array size is also reduced and the number of hash functions are increased. The false positivity rate is reduced and time taken to add all words also reduced to 40 mins on the system.
For part 2 of the question, we are required to experimentally find out the values of n and k for a false positivity rate of 0.8.
For this, I take a small subset of the streaming data given, but large enough to notice the difference. I keep a subset of 50k words and run 2 for loops for the n value and k value. The following is the result -
For the sample csv -

```
number of hash functions 1
bit array size 9289
Number of False Positives:  40739
Number of Lookup Count:  40750
Time taken for n = 20000 and k = 1 is 333.77546644210815
number of hash functions 2
bit array size 9289
Number of False Positives:  40739
Number of Lookup Count:  40750
Time taken for n = 20000 and k = 2 is 317.61678862571716
number of hash functions 3
bit array size 9289
Number of False Positives:  40739
Number of Lookup Count:  40750
Time taken for n = 20000 and k = 3 is 320.09696769714355
number of hash functions 4
bit array size 9289
Number of False Positives:  40739
Number of Lookup Count:  40750
Time taken for n = 20000 and k = 4 is 313.93130469322205
```

```
number of hash functions 1
bit array size 13933
Number of False Positives:  36444
Number of Lookup Count:  36455
Time taken for n = 30000 and k = 1 is 262.3304355144501
number of hash functions 2
bit array size 13933
Number of False Positives:  36444
Number of Lookup Count:  36455
Time taken for n = 30000 and k = 2 is 258.2802519798279
number of hash functions 3
bit array size 13933
Number of False Positives:  36444
Number of Lookup Count:  36455
Time taken for n = 30000 and k = 3 is 277.62720346450806
number of hash functions 4
bit array size 13933
Number of False Positives:  36444
Number of Lookup Count:  36455
Time taken for n = 30000 and k = 4 is 282.3213403224945
```

```
number of hash functions 1
bit array size 18578
Number of False Positives:  32663
Number of Lookup Count:  32674
Time taken for n = 40000 and k = 1 is 262.87142157554626
number of hash functions 2
bit array size 18578
Number of False Positives:  32663
Number of Lookup Count:  32674
Time taken for n = 40000 and k = 2 is 270.2783923149109
number of hash functions 3
bit array size 18578
Number of False Positives:  32663
Number of Lookup Count:  32674
Time taken for n = 40000 and k = 3 is 262.28436255455017
number of hash functions 4
bit array size 18578
Number of False Positives:  32663
Number of Lookup Count:  32674
Time taken for n = 40000 and k = 4 is 293.75014781951904
```

```
number of hash functions 1
bit array size 23222
Number of False Positives:  29418
Number of Lookup Count:  29429
Time taken for n = 50000 and k = 1 is 273.63077569007874
number of hash functions 2
bit array size 23222
Number of False Positives:  29418
Number of Lookup Count:  29429
Time taken for n = 50000 and k = 2 is 273.34508752822876
number of hash functions 3
bit array size 23222
Number of False Positives:  29418
Number of Lookup Count:  29429
Time taken for n = 50000 and k = 3 is 288.89156794548035
number of hash functions 4
bit array size 23222
Number of False Positives:  29418
Number of Lookup Count:  29429
Time taken for n = 40000 and k = 4 is 284.79937648773193
```

Thus we can see that the