ECE 590 HW 5

Chelsea (Jiyuan) Lyu

Nov. 7 2023

1 Problem 1

Here is my function which removes duplicates from the data:

```
1 def rmdup(data):
2     unique = {}
3     last_occur = []
4
5     for element in reversed(data):
6         if element not in unique:
7             last_occur.append(element)
8             unique[element] = True
9
10     last_occur.reverse()
11    return last_occur
```

Here is the O of runtime for each line.
N is the input data size.

Line	How Many Times?	How Long?
1	$O(1)$	$O(1)$
2	$O(1)$	$O(1)$
3	$O(1)$	$O(1)$
4		
5	$O(N)$	$O(1)$
6	$O(N)$	$O(1)$
7	$O(N)$	$O(1)$
8	$O(N)$	$O(1)$
9		
10	$O(1)$	$O(N)$
11	$O(1)$	$O(1)$

Runtime is $O(1+1+1+N+N+N+N+N+1) = O(N)$

Here I measure the runtime for various data sizes three times and calculate the average. The average runtime is shown below in Graph 1 (Figure 1) and Table 2 (Figure 2).

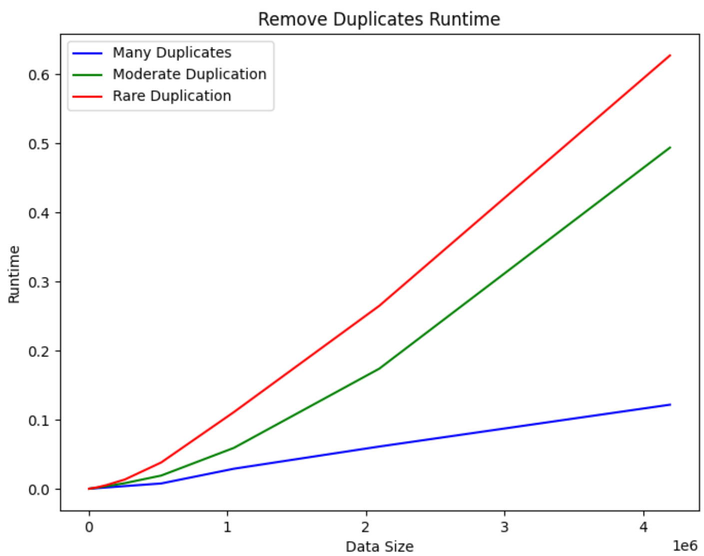

Figure 1: Graph 1

	Many Duplicates	Moderate Duplication	Rare Duplication
4096	0.00010633468627929688	7.54992167154948e-05	0.00012882550557454428
8192	0.00016967455546061197	0.00022514661153157553	0.0003064473470052083
16384	0.00028403600056966144	0.00045800209045410156	0.0005166530609130859
32768	0.0005200703938802084	0.0008426507314046224	0.0010985533396402996
65536	0.0009990533192952473	0.0018026034037272136	0.0022416114807128906
131072	0.001950661341349284	0.003766934076944987	0.005450328191121419
262144	0.003899256388346354	0.008163849512736002	0.013484875361124674
524288	0.0077250003814697266	0.019172509511311848	0.03812837600708008
1048576	0.02904669443766276	0.05922492345174154	0.11102739969889323
2097152	0.06116334597269694	0.17386269569396973	0.26482899983723956
4194304	0.12173223495483398	0.4938444296518962	0.6272506713867188

Figure 2: Table 1

According to Graph 1 and Table 1, with the data size increase, the runtime increase linear, which match my theory runtime $O(N)$.

However, there are different behaviors for different categories of input. When there are fewer duplicates in the input, the runtimes increase faster when data size increases. This might be due to fewer duplicates leading to more elements needing to be stored in the output array.

2 Problem 2

Here is my function which multiplies the matrix:

```
1 def matrix_mul(a,b):
2     c = []
3     for i in range(len(a)):
4         row = []
5         for j in range(len(b[0])):
6             element = 0
7             for k in range(len(b)):
8                 element += a[i][k] * b[k][j]
9             row.append(element)
10        c.append(row)
11    return c
```

Here is the O of runtime for each line.
Let the input matrix a be $X * Y$ and the input matrix b be $Y * Z$.

Line	How Many Times?	How Long?
1	$O(1)$	$O(1)$
2	$O(1)$	$O(1)$
3	$O(X)$	$O(1)$
4	$O(X)$	$O(1)$
5	$O(X * Y)$	$O(1)$
6	$O(X * Y)$	$O(1)$
7	$O(X * Y * Z)$	$O(1)$
8	$O(X * Y * Z)$	$O(1)$
9	$O(X * Y)$	$O(1)$
10	$O(X)$	$O(1)$
11	$O(1)$	$O(1)$

Runtime is $O(1 + 1 + X + X + X * Y + X * Y + X * Y * Z + X * Y * Z + X * Y + X + 1) = O(X * Y * Z)$

Since three categories of the shape of the input matrix are $(N * 4) \times N * N \times (N/4)$ for many rows by few columns, $N \times N * N \times N$ for squares, and $(N/4) \times N * N \times (N * 4)$ for few rows by many columns, the runtime $O(X * Y * Z) = N^3$ for three shapes.

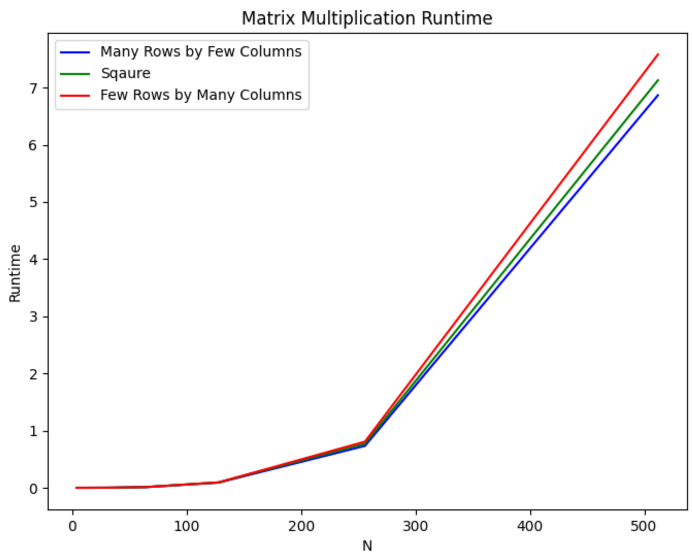

Figure 3: Graph 2

	Many Rows by Few Columns	Sqaure	Few Rows by Many Columns
4	8.58306884765625e-06	5.8015187581380206e-06	4.8478444417317706e-06
8	3.298123677571615e-05	3.0358632405598957e-05	2.9722849527994793e-05
16	0.00021505355834960938	0.0001998742421468099	0.00020043055216471353
32	0.0014580885569254558	0.001492897669474284	0.0014473597208658855
64	0.011162598927815756	0.01136922836303711	0.011335372924804688
128	0.08977333704630534	0.0938723882039388	0.09366575876871745
256	0.733262300491333	0.7703099250793457	0.8071232636769613
512	6.867247263590495	7.128139972686768	7.580312172571818

Figure 4: Table 2

Here I measure the runtime for various data sizes three times and calculate the average. The average runtime is shown above in Graph 2 (Figure 3) and Table 2 (Figure 4).

According to Graph 2 and Table 2, with the data size increase, the runtime increases match my theory runtime $O(N^3)$. The lines do not look like curves due to there are not enough different data sizes and runtimes.

The behaviors for different categories of matrix shapes are quite similar. But we can observe that when there are fewer rows, it'll take slightly more runtime, which could be due to the ignoring part $O(X)$ in the former calculation.

3 Problem 3

Here is my function which matches the length of the substrings:

```
def matching_length_sub_strs(s, c1, c2):
    c1_dict = {}
    c2_dict = {}
    ans = set()

    count1 = 0
    count2 = 0
    for i in range(len(s)):
        if s[i] == c1:
            count1 = count1 + 1
        if s[i] != c1 or i == len(s)-1:
            index1 = i - count1
            if i == len(s)-1:
                index1 = i - count1 + 1
            if count1 != 0:
                if count1 in c1_dict:
                    c1_dict[count1].append(index1)
                else:
                    c1_dict[count1] = [index1]
                count1 = 0

        if s[i] == c2:
            count2 = count2 + 1
        if s[i] != c2 or i == len(s)-1:
            index2 = i - count2
            if i == len(s)-1:
                index2 = i - count2 + 1
            if count2 != 0:
                if count2 in c2_dict:
                    c2_dict[count2].append(index2)
                else:
                    c2_dict[count2] = [index2]
                count2 = 0
    for key in c1_dict:
        if key in c2_dict:
            for c1_value in c1_dict[key]:
                for c2_value in c2_dict[key]:
                    tuple = (c1_value, c2_value, int(key))
                    ans.add(tuple)
    return ans
```

Here is the O of runtime for each line.
N is the length of the input string.

Line	How Many Times?	How Long?
1	$O(1)$	$O(1)$
2	$O(1)$	$O(1)$
3	$O(1)$	$O(1)$
4	$O(1)$	$O(1)$
5	$O(1)$	$O(1)$
6	$O(1)$	$O(1)$
7	$O(1)$	$O(1)$
8	$O(N)$	$O(1)$
9	$O(N)$	$O(1)$
10	$O(N)$	$O(1)$
11	$O(N)$	$O(1)$
12	$O(N)$	$O(1)$
13	$O(N)$	$O(1)$
14	$O(N)$	$O(1)$
15	$O(N)$	$O(1)$
16	$O(N)$	$O(1)$
17	$O(N)$	$O(1)$
18	$O(N)$	$O(1)$
19	$O(N)$	$O(1)$
20	$O(N)$	$O(1)$
21	$O(N)$	$O(1)$
22	$O(N)$	$O(1)$
23	$O(N)$	$O(1)$
24	$O(N)$	$O(1)$
25	$O(N)$	$O(1)$
26	$O(N)$	$O(1)$
27	$O(N)$	$O(1)$
28	$O(N)$	$O(1)$
29	$O(N)$	$O(1)$
30	$O(N)$	$O(1)$
31	$O(N)$	$O(1)$
32	$O(N)$	$O(1)$
33	$O(N)$	$O(1)$
34	$O(N)$	$O(1)$
35	$O(N)$	$O(1)$
36	$O(N^2)$	$O(1)$
37	$O(N^3)$	$O(1)$
38	$O(N^3)$	$O(1)$
39	$O(N^3)$	$O(1)$
40	$O(1)$	$O(1)$

According to the table of Big O above, the runtime is $O(N^3)$

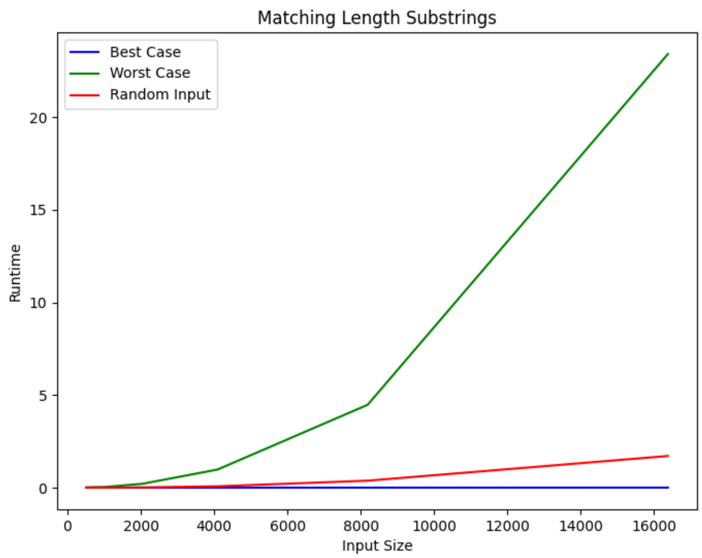

Figure 5: Graph 3

	Best Case	Worst Case	Random Input
512	9.846687316894531e-05	0.01827685038248698	0.0013113021850585938
1024	0.00018866856892903647	0.037659645080566406	0.004580736160278320
2048	0.000383297602335612	0.21045629183451334	0.012797196706136068
4096	0.0007717609405517578	0.9798628489176432	0.07353417078653972
8192	0.001546303431193034	4.4758516152699785	0.3780324459075928
16384	0.003115971883138021	23.415177822113037	1.7102687358856201

Figure 6: Table 3

Here are three cases of the input:

1. The best case is that the input string only has $c1 = a$ or $c2 = b$. Here I assume there only exists $c1 = a$ in the string, so the regular expression of best case is a^N, and N is the length of the string.

2. The worst case is that the input string does not have a contiguous character $c1 = a$ or $c2 = b$, so the regular expression of the worst case is $(ab)^{N/2}$, and N is the length of the string.

3. The random input is a string of length N which is approximately 3/7 "a"s, 3/7 "b"s, and 1/7 a capital letter.

I measure the runtime for various data sizes three times and calculate the average. The average runtime is shown above in Graph 3 (Figure 5) and Table 3 (Figure 6).

According to Graph 3 and Table 3, with the data size increase, the runtimes increase matching my theory runtime $O(N^3)$.

4 Problem 4

(a) T

(b) T

(c) T

(d) F

(e) F