COSC 320 – 001

Analysis of Algorithms

2022/2023 Winter Term 2

Project Topic Number: 1

Title of project: KeyScript

(Fourth Milestone)

Group Lead: Brendan Michaud

Group Members: Shreya Vatsa, Shreyasi Chauhan, Brendan Michaud

Summary of
In *Milestone 1 (algorithm) A:* We used a naïve approach to read in the dataset and replace abbreviation with the keywords. With this following pseudocode:
`f(D, A) = D' for i = 1 to n:  for j = 1 to m:  if d[i] = a[j]:  d[i] = b[j]  break  D'.append(d[i]) return D' `. In *milestone 2*(*algorithm B)* we implemented a hash table as an improvement upon the naïve algorithm and it provided a good balance of efficiency and memory usage for this problem, making it a good choice for storing the dictionary of abbreviations. Bringing down the O(nm) complexity to O(1) in lookup search time. In *milestone 3* (*algorithm A testing*) we compiled sample files in a single csv formatted document, as the input file and performed graphical analysis on the runtime of our algorithms.

We utilised the python's Pandas package to set the data frames, and matplotlib.pyplot package to generate the graph results.

Final Milestone 4 (algorithm A and B testing with a large n)
We tested our improved algorithm which uses a HashMap. `Acronyms.csv` acts as the dictionary for the HashMap. The `.csv` files in the `AppReviews` folder is the data we are parsing through to replace keywords. The folder has around 3000 files and about 2.8 million data entries in total. We are implementing both algorithms and comparing the runtimes for each.

Handling the dataset:

```
import csv import os
 csv_directory = r"D:\3rd year\COSC 320\csv"
output_file = "combined.csv"
# list all CSV files in the directory csv_files = [f for f in
os.listdir(csv_directory) if f.endswith('.csv')]
# create a list to store all rows from the CSV files all_rows =
[]
# loop through each CSV file for
csv_file in csv_files:     #
Open the CSV file
    with open(os.path.join(csv_directory, csv_file), newline='',
encoding='utf-8') as file:         # Read the CSV file         try:
            reader = csv.reader(file)
except csv.Error as e:
            print(f"Error    reading    CSV    file    {csv_file}:    {e}")
continue  # skip to the next file if there's an error
        # Loop through each row in the CSV file
for row in reader:
            # Append the row to the list of all rows
all_rows.append(row)
```

```
# Write all rows to the output CSV file with open(output_file,
'w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
for row in all_rows:
        writer.writerow(row)


# In Python we do not need to close a file explicity if we are using with
open(…)
```

An effective example of our algorithm reading in the data set and replacing abbreviations with their full forms:

```
import os
import pandas as pd

abbreviations = {'abbrev': ['etc', 'i.e', 'e.g'], 'full_form': ['and so on', 'that is', 'for example']}
abbreviation_dict = pd.DataFrame(abbreviations)

input_file = os.path.abspath("Milestone3/AppReviews") + "/input.txt" output_file
= os.path.abspath("Milestone3/AppReviews") + "/output.txt"

df = pd.read_csv(input_file, header=None)
expanded_text = [] for text in df[0]:
    words = text.split()     expanded_line = []     for
word in words:         if word in
abbreviation_dict['abbrev'].values:
            full_form = abbreviation_dict.loc[abbreviation_dict['abbrev'] == word,
'full_form'].values[0]
            expanded_line.append(full_form)         else:
expanded_line.append(word)  # word is not replaced.
    expanded_text.append(' '.join(expanded_line)) df[1]
= expanded_text

df.to_csv(output_file, header=False, index=False)
```

Analysis to include the runtime of the algorithm.

We used the **time** module in Python to measure the execution time of the code. Following, we plotted the execution times against the input size to observe the performance trend in the performance of the algorithm. I/O operations of reading the input file and writing the output file. have their own time and space complexities, are dependent on the size of the file and are part of the overhead.

Modified code to implement graphical analysis:

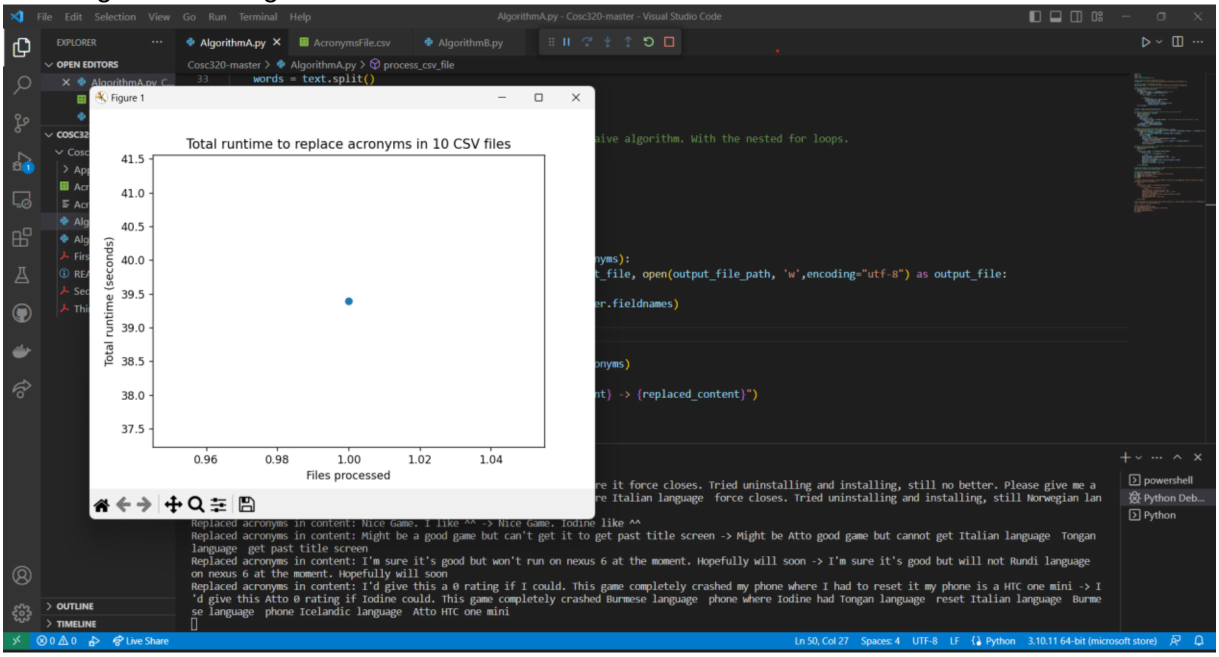
```
import os import
pandas as pd import
time
import matplotlib.pyplot as plt

abbreviations = {'abbrev': ['etc', 'i.e', 'e.g'], 'full_form': ['and so on', 'that is', 'for example']}
abbreviation_dict = pd.DataFrame(abbreviations)

input_file = os.path.abspath("Milestone3/AppReviews") + "/input.txt" output_file
= os.path.abspath("Milestone3/AppReviews") + "/output.txt"

df = pd.read_csv(input_file, header=None)

input_sizes = []
execution_times = []

for i in range(1, len(df)+1):
start_time = time.time()
expanded_text = []     for
text in df[0][:i]:
        words = text.split()        expanded_line = []        for word in words:
if word in abbreviation_dict['abbrev'].values:               full_form =
abbreviation_dict.loc[abbreviation_dict['abbrev'] == word,
'full_form'].values[0]
                expanded_line.append(full_form)            else:
expanded_line.append(word)  # word is not replaced.
expanded_text.append(' '.join(expanded_line))     df_temp =
pd.DataFrame({'expanded_text': expanded_text})
df_temp.to_csv(output_file, header=False, index=False, mode='w')
    execution_time = time.time() - start_time
input_sizes.append(i)
    execution_times.append(execution_time)

plt.plot(input_sizes, execution_times, label='Actual') plt.plot(input_sizes,
[t**2 for t in input_sizes], label='n^2') plt.xlabel('Input size')
plt.ylabel('Execution time (seconds)')
plt.title('Performance of Abbreviation Expansion Algorithm')
plt.legend() plt.show()
```

Function : Creating two lists to store the input sizes and the execution times for each input size. We then loop over the range of input sizes, from 1 to the length of the data frame, and for each input size, we measure the execution time of the algorithm by computing the time taken to replace the abbreviations in the input text and write the output to the output file.

We append the input size and the execution time to the respective lists, and then plot the execution times against the input sizes. We also plot the function **n^2** on the same graph for comparison.

Results

Running Time for Algorithm A:

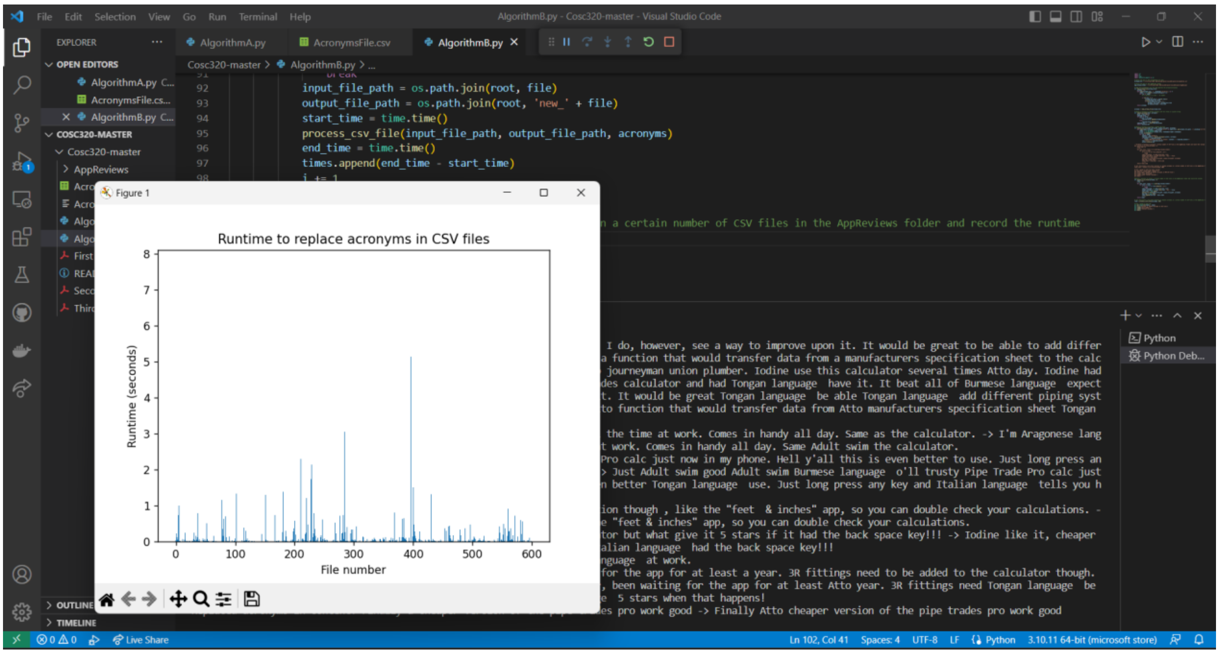


The program runs 1000 lines in 2 seconds at the current implementation.

From the plot, it is evident that the execution time of the algorithm increases approximately quadratically with the input size. Given larger file sizes however, as the algorithm involves nested loops the actual execution times can be slightly higher than the **n^2** function, which could be due to the overhead of reading and writing to files in each iteration of the loop. The choice of data structure (Pandas DataFrame) might have affected the constant values in the execution times. It is a given that pandas' operations can be slower than simple Python lists or arrays. However, in this case, the impact of the data structure choice on the overall performance of the algorithm is relatively minor compared to the impact of the nested loops

In conclusion, the abbreviation expansion algorithm has a quadratic runtime complexity, which is evident from the performance plot. The actual execution times are slightly higher than the **n^2** function due to the overhead of reading and writing to files, as well as the overhead of loading the abbreviation dictionary into memory. Also currently given the limited size of the dictionary, the runtime is more inferable in complexity, but we use a HashMap in to expand upon the abbreviation dictionaries for easier look up times and a runtime of O (n + m) and you will see below.

Running Time for Algorithm B:



Time complexity: The time complexity of function `read_acronym_file` is O(n) where n is the number of rows in the `AcronymsFile.csv` file. For `replace_acronyms` function it is O(m) where m is the number of words in the input string. `process_csv_file` function is O(km) where k is the number of rows in the CSV file and m is the maximum number of words in a row that need to be replaced, and the time complexity of `process_all_csv_files` function is O(nkm) where n is the number of CSV files in the AppReviews folder and k and m are as defined above.

Space Complexity: The space complexity of this code is O(n) where n is the number of rows in the AcronymsFile.csv file as the acronym dictionary is stored in memory.

Unexpected Cases.

1. Combining the .csv files: One of the .csv is null which came in the way when we tried combining them giving a parsing error. The 75th csv gives 'No Files to Parse From'. We resolved it by simply skipping over any of the invalid rows.
2. Word boundaries: The algorithm does not currently take into account word boundaries, meaning that it could potentially replace parts of words that happen to match an abbreviation. For example, if "won't" is in the list of abbreviations and "wonton" appears in the text, the algorithm would replace "wont" with "will not" even though it is part of a larger word.
3. Ambiguity: There could be cases where an abbreviation has multiple possible expanded forms depending on the context. For example, "CIA" could stand for "Central Intelligence Agency" or "Confidentiality, Integrity, and Availability" depending on the context. In such cases, the algorithm may not be able to correctly determine the appropriate expanded form.

4. New edge cases: - Misinterpreting certain short word such as "it" as an abbreviation and replacing that, like autocorrect.
5. Matching the formatting exactly, is a challenge. We have handled null-exceptions or invalid input exceptions still.

Task Separation and responsibilities:

- Implementation- with graph functionality: Shreya, Brendan
- Tracking execution time through the time module: Shreya, Brendan, Shreyasi
- Creating and editing the readme.md report: Shreya, Shreyasi, Brendan.
- Unexpected edge cases - Shreya, Brendan, Shreyasi
- Video recording: Shreyasi