Ahmad Shah, Neeti Mistry, Sara Gaber, Sohan Chatterjee
Professor AlOmar
SSW 567
6 December 2024

PerfTesting

1. GitHub Repository

https://github.com/neetixmistry/SSW567-FinalProject/tree/main/PerfTesting

2. Introduction

This report analyzes the execution time of encoding and decoding passport records using two sets of data—one containing encoded records and the other containing decoded records. The performance is measured in two scenarios:

1. Execution time without tests — measuring the raw time to process the data
2. Execution time with unit tests — including unit tests for both encoding and decoding to evaluate the impact of testing on performance

3. PerfTest.py

```
import time
import json
import sys
import unittest
import csv
from unittest.mock import patch

sys.path.insert(0, 'UnitTesting')
from MRTD import MRTDProcessor
from MTTDtest import TestMRTDProcessor
processor = MRTDProcessor()
unit_tests = TestMRTDProcessor()


##get encoded data
encoded_data = []
with open('PerfTesting/records_encoded.json', 'r') as file:
    raw_data = json.load(file)
raw_data = raw_data["records_encoded"]
```

```
for x in raw_data:
    encoded_data += [x.split(";")]

##get decoded data
decoded_data = []
with open('PerfTesting/records_decoded.json', 'r') as file:
    raw_data = json.load(file)
raw_data = raw_data["records_decoded"]
for x in raw_data:
    decoded_data += [x["line1"] | x['line2'] | {'passport_type': "P"}]


def run_performance_test(n, decode):
    initial_time = time.perf_counter()
    if (decode):
        my_data = decoded_data[0:n]
        for x in my_data:
            processor.encode_mrz(x)
    else:
        my_data = encoded_data[0:n]
        for x in my_data:
            processor.decode_mrz(x)
    final_time = time.perf_counter()
    return final_time - initial_time

def run_unitTest_performance_test(n, decode):
    my_decode_data = decoded_data[0:n]
    my_encode_data = encoded_data[0:n]
    encode_time = 0
    decode_time = 0
    for x in range(n):
        unit_tests.processor = processor
        unit_tests.test_fields = my_decode_data[x]
        unit_tests.mrz_lines = my_encode_data[x]

        initial_time = time.perf_counter()
        unit_tests.test_valid_decode_mrz()
        final_time = time.perf_counter()
        decode_time += (final_time - initial_time)
```

```
        initial_time = time.perf_counter()
        unit_tests.test_encode_mrz()
        final_time = time.perf_counter()
        encode_time += (final_time - initial_time)
    if(decode):
        return encode_time
    else:
        return decode_time

def create_results(filepath, decode):
    data = [["Lines Read", "Execution time without tests", "Execution time with unit tests"]]
    kList = [100] + list(range(1000, 10001, 1000))
    for k in kList:
        execution = run_performance_test(k, decode)
        tests = run_unitTest_performance_test(k, decode)
        results = [k*2] + [execution] + [execution + tests]
        data += [results]
    with open(filepath, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerows(data)

create_results('PerfTesting/PerfTesting_DecodeData_Results.csv', True)
print("results created for decode data")
create_results('PerfTesting/PerfTesting_EncodeData_Results.csv', False)
print("results created for encode data")
```

4. Methodology

The performance was tested by processing subsets of the data (ranging from 200 to 20,000 lines in increments of 2,000) for both encoding and decoding scenarios. The data was loaded from two JSON files:

1. records_encoded.json: Contains encoded records in the form of MRTD (Machine Readable Travel Document) data
2. records_decoded.json: Contains decoded data corresponding to the encoded records

The execution times were measured using Python's time.perf_counter() function for both cases (with and without unit tests). The unit tests were implemented to verify the functionality of the encoding and decoding functions.

5. Results

The following graphs represent the execution times for processing different numbers of records, with and without unit tests.

For Decoding Data:

- As the number of records increases, the execution time without tests increases linearly.
- The execution time with unit tests grows more significantly due to the additional validation and testing steps for each record.

For Encoding Data:

- Similar trends are observed. The execution time without tests shows a steady increase, but with unit tests, the time grows more quickly due to the added test executions.

Here are the detailed results for both encoding and decoding:

PerfTesting_DecodeData_Results

Lines Read	Execution time without tests	Execution time with unit tests
200	0.0016191000000000100	0.0037409000000000700
2000	0.015943100000000000	0.03572560000000080
4000	0.03747910000000000	0.07825589999999930
6000	0.047562400000000000	0.10249270000000000
8000	0.06293990000000010	0.15538930000000200
10000	0.08037520000000010	0.17667220000000400
12000	0.10088920000000000	0.22975379999998800
14000	0.1254822	0.28473739999999500
16000	0.18401900000000000	0.3646637000000020
18000	0.1708151	0.423406700000001
20000	0.2645021000000000	0.5809810999999990

PerfTesting_EncodeData_Results

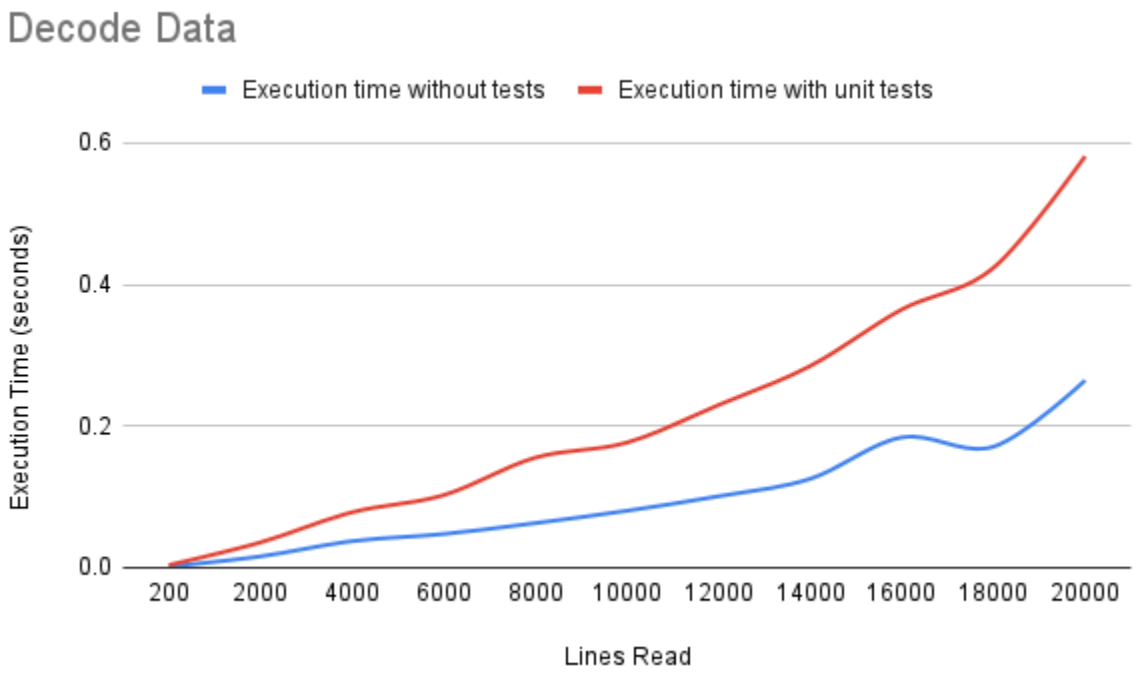
Lines Read	Execution time without tests	Execution time with unit tests
200	0.0005348999999998940	0.006196699999994950
2000	0.005033499999999690	0.04905130000000300
4000	0.005471000000000000	0.09172919999998990
6000	0.009922999999999680	0.13530549999999600
8000	0.011329599999999800	0.23731219999998500
10000	0.014379399999999300	0.24113280000001300
12000	0.017140800000000000	0.29364829999997300
14000	0.022366300000000700	0.3146715999999990
16000	0.02172760000000020	0.3633608999999410
18000	0.028997200000000100	0.4046012999999980
20000	0.027308699999999000	0.4445666000000620

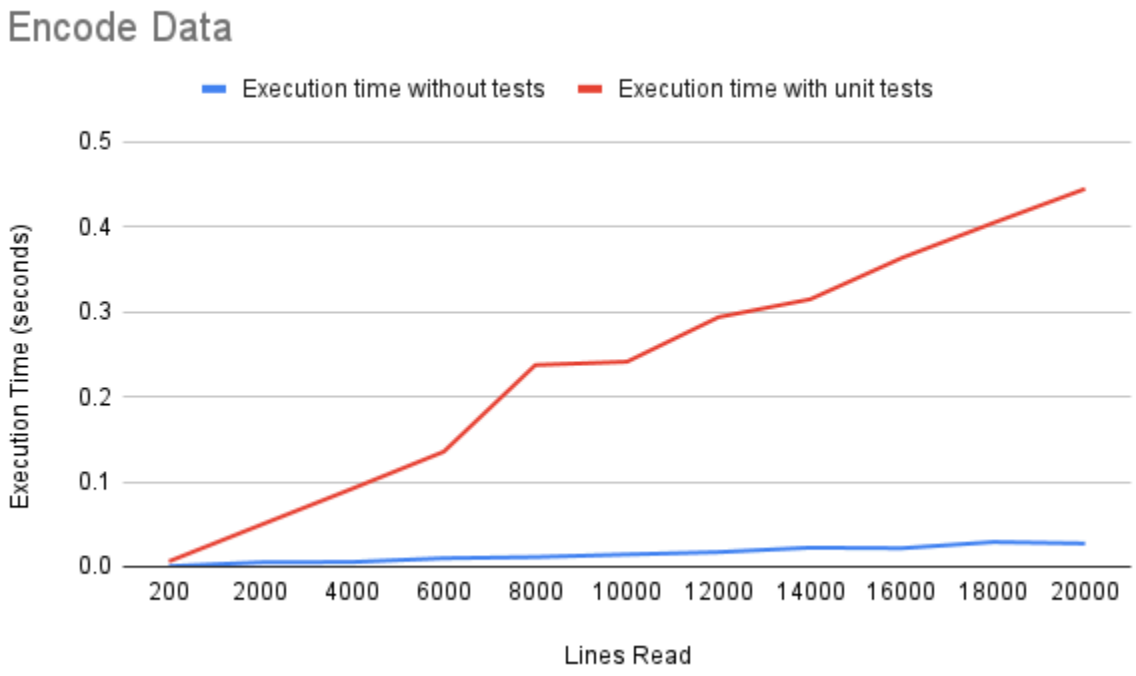
6. Analysis

- Impact of Unit Testing
    - The results show that adding unit tests significantly increases execution time. This is expected because each record requires additional validation during processing.
    - For example, with 2,000 lines, the decoding time without tests is 0.0159 seconds, but with unit tests, it increases to 0.0357 seconds. This gap becomes more significant as the number of records increases.
- Execution Time Growth
    - Both the encoding and decoding operations show a nearly linear increase in execution time without tests, suggesting that the time complexity for both processes is roughly O(n).
    - The increase in execution time with unit tests is more noticeable due to the overhead of performing assertions for each record.
- Use Case Considerations
    - The results suggest that for performance-critical applications, it might be better to avoid unit testing in the production environment, especially with large datasets.
    - However, unit tests are crucial for verifying correctness, and the performance penalty is often acceptable during development or in smaller datasets.

7. Plot of Results

7.1 Line Graphs





7.2 Explanation

The graphs demonstrate the effect of unit tests on the execution time for both encoding and decoding. As the number of records increases, both curves show a steady increase in execution time. The line for execution time without tests increases linearly, suggesting that processing time scales with the size of the dataset. However, the line for execution time with unit tests grows more steeply, indicating that the overhead introduced by testing significantly impacts performance, especially with larger datasets. This illustrates the trade-off between ensuring correctness through testing and the added performance cost, which is particularly relevant for large-scale applications.

7.3 Google Sheets URL

https://docs.google.com/spreadsheets/d/1ySttZFi4vZNCRL0MVaDWHKlewpit1Na4mV_TzVwiqMk/edit?usp=sharing

8. Conclusion

The results clearly demonstrate the trade-off between validation (via unit tests) and raw execution time. While unit tests increase execution time, they are essential for ensuring the correctness of the encoding and decoding functions. Depending on the use case, it might be necessary to balance the need for testing and performance, especially when processing large datasets like the passport records in this study.

9. Deliverables

- The CSV files with the measured execution times:
    - PerfTesting_DecodeData_Results.csv
    - PerfTesting_EncodeData_Results.csv
- The source code (PerfTest.py) used to run the experiments is available in the provided repository link: https://github.com/neetixmistry/SSW567-FinalProject
- Google sheets plots containing the measurements and graphs