HO CHI MINH INTERNATIONAL UNIVERSITY

PROJECT REPORT

Data Structures & Algorithms

Project Topic: Bank Statement Inquiry Software

Team Members:

ID	Name	Contribution
ITITWE22101	Nguyễn Nhật Huy	50%
ITITWE22143	Lê Nhật Duy	50%

1. INTRODUCTION

Build a web/app to look up statement data of the “Mat Tran To Quoc”. The application needs an interface (web/app) to look up statement information by: amount, sender name, content.

2. LANGUAGE

2.1 Python

Python is well suited for searching through CSV files. The number of records is a relatively significant amount of data, but Python's libraries can handle it efficiently. Here are some library of Python that can help in this project:

- **CSV**: is a Python built-in library that works with raw CSV data without loading the entire dataset into memory by reading the data line by line. This method uses less memory but can be slower if searching for multiple conditions or performing complex queries.
- **Pandas**: is a great library for working with large datasets. It can quickly read CSV files into a DataFrame, allowing for quick data manipulation and searching. Pandas can handle data with millions of rows and supports optimizations such as use to improve search performance.
- **SQLite**: for more advanced search functionality and optimization, load CSV data into a SQLite database. SQLite is lightweight and allows for fast SQL queries.

2.2 Library

Data lookup needs to be done quickly and efficiently. Pandas library is an optimal choice.

- Load data: CSV file is loaded into a global variable ***data_df*** using ***pandas.read_csv***, UTF-8 encoding and Byte-order Mark (BOM) handling (utf-8-sig).

The OS module provides a way to interact with the operating system, such as reading file paths, directories, and other operating system-level functions.

- ***os.path.dirname*** and ***os.path.abspath*** find the absolute path of the current directory and work with the ***index.html*** file correctly.

Flask is a lightweight web framework used to build web applications. In this project, **Flask** is used to create API endpoints, handle HTTP requests, process data from CSV files, and return results to the user. Flask acts as the “*backbone*” of the web server, managing requests from users and returning appropriate responses.

- **Flask**: core class, used to initialize the application
- ***request***: handle HTTP request data (e.g. query parameters)
- ***jsonify***: convert Python objects (like dictionaries) to JSON data type
- ***send_from_directory***: serve static files, in this case ***index.html***

3. DATA STRUCTURES

A CSV (Comma Separated Values) file is a text file that contains data in a structured format, where each value is separated by a comma. Each line represents a record, and each value in the line corresponds to a field in the record.

Ex: Data from CSV file ***date_time***, ***trans_no***, ***credit***, ***debit***, ***detail*** are headers.

```
date_time , trans_no , credit ,debit , detail
03/09/2024 _5216 .65140 ,9 ,200000 ,0 ,120167.030924.100642. NGUYEN
HOAI NAM ung ho
03/09/2024 _5017 .43849 ,13 ,9000 ,0 ,888828.030924.121121. NGUYEN THI
KIEU OANH chuyen tien
04/09/2024 _5240 .80637 ,55 ,500000 ,0 , MBVCB .6944619812. chung tay
gop suc .CT tu 0351000801710 TRAN DANH VU toi 0011001932418
MAT TRAN TO QUOC VN - BAN CUU TRO TW
```

- Pandas provides a simple way to read and manipulate CSV files: ***pandas.read_csv***.
- The user submits search parameters in ***index.html***. **Flask** processes these parameters, applies **Pandas** filtering logic, and returns the results as JSON.
- **JavaScript** takes the JSON response and dynamically builds an HTML table to display the filtered results.

  Data returns as:

```
{
" date_time ": " 03/09/2024 _5216 .65140 ",
" trans_no ": "9",
" credit ": " 200000 ",
" debit ": "0",
" detail ": " 120167.030924.100642. NGUYEN HOAI NAM ung ho"
},
{
" date_time ": " 03/09/2024 _5017 .43849 ",
" trans_no ": "13",
" credit ": " 9000 ",
" debit ": "0",
" detail ": " 888828.030924.121121. NGUYEN THI KIEU OANH chuyen
tien "
},
{
" date_time ": " 04/09/2024 _5240 .80637 ",
```

" trans_no ": "55",
" credit ": " 500000 ",
" debit ": "0",
" detail ": " MBVCB .6944619812. chung tay gop suc .CT tu 0351000801710 TRAN DANH VU toi 0011001932418 MAT TRAN TO QUOC VN - BAN CUU TRO TW"
}

4. ALGORITHMS

Algorithm for filtering **DataFrame** (***data_df***) using **Pandas**. This algorithm applies filter conditions on the data and returns the filtered results as a dictionary list.

1. Input validation: this function checks if ***data_df*** is ***None***: as if, it means that the dataset has not been loaded and the function will immediately return an empty list (**[ ]**).
2. Create a copy of the **DataFrame**: a copy of ***data_df*** is created and saved to ***filtered_df***. This ensures that the original dataset is not changed when the data filtering processes are performed.
3. Repeat the filtering conditions:
    - ***credit***: numeric range or an exact match.
    - ***date_time***: containing the match - check if value is a substring of ***date_time*** using ***str.contains.na=False*** to ensure that rows with ***NaN*** in the column are eliminated.
    - ***trans_no*** : containing the match - ensure value is a numeric data type.
    - Other columns: case-insensitive string comparison using ***str.containswith*** and ***case=False***. For example, "Python" matches either "python" or "PYTHON".

```
# Filter data using pandas
def filter_data(conditions):
  global data_df

  if data_df is None:
    return []

  filtered_df = data_df.copy()

  for column, value in conditions.items():
    if column == 'credit':
      if isinstance(value, tuple): # Range query
```

```
        min_value, max_value = value
        filtered_df =
filtered_df[(filtered_df['credit'] >= min_value) &
(filtered_df['credit'] <= max_value)]
      else: # Single value
        filtered_df = filtered_df[filtered_df['credit']
== value]
    elif column == 'date_time':
      filtered_df =
filtered_df[filtered_df['date_time'].str.contains(value
, na=False)]
    elif column == 'trans_no':
      try:
        value = int(value) # Ensure the input is
numeric
        filtered_df =
filtered_df[filtered_df['trans_no'] == value]
      except ValueError:
        # If value is not a valid number, skip
filtering this column
        continue
    else:
      filtered_df =
filtered_df[filtered_df[column].str.contains(value,
na=False, case=False)]

  return filtered_df.to_dict(orient='records')
```

4.1 Filtering (Search Algorithm)

- **Linear Search**:
  - For filtering rows in the `filter_data()` function, the code applies conditions column by column.

- Filtering a Pandas DataFrame by value(s) (`filtered_df[filtered_df['credit'] >= min_value]`) essentially loops over rows (though implemented in C for optimization).
- **Time Complexity**:
  - Each filtering operation over `n` rows has a complexity of **O(n)**.
  - Multiple filter conditions are combined, leading to sequential passes over the dataset.

4.2. Range Query

When querying the `credit` field, the algorithm checks for a range:

```
filtered_df = filtered_df[(filtered_df['credit'] >= min_value) & (filtered_df['credit'] <= max_value)]
```

- This is a linear pass through the DataFrame, where each row is checked to see if its value falls within the specified range.
- **Time Complexity**: **O(n)** for a DataFrame with `n` rows.

4.3. String Matching

For columns like `date_time` and other string-based columns, the code uses string matching to filter rows:

```
filtered_df = filtered_df[filtered_df[column].str.contains(value, na=False, case=False)]
```

- This involves searching for substrings within each cell of the column.
- The underlying implementation in Pandas uses vectorized string operations, but conceptually, it's similar to:
  - **Substring Search Algorithms**:
    - Naive search: **O(m * n)** where `m` is the length of the query string, and `n` is the length of the text in each cell.
    - However, Pandas often uses more optimized techniques (e.g., SIMD-based implementations).

4.4. Validation Algorithms

- **Date Format Validation**: The code validates the format of `date_time` values using a **regex pattern**:

```
if not pd.Series([value]).str.match(r'\d{2}/\d{2}/\d{4}').bool():
return jsonify({"error": "Invalid date_time format. Use DD/MM/YYYY."}), 400
```

- **Time Complexity**: Regex matching has a complexity that depends on the regex engine and input size (average case is linear).
- **Credit Value Validation**: Validates numeric ranges:

```
min_value, max_value = map(float, value.split('-'))
if min_value > max_value:
  return jsonify({"error": "Invalid range: min_value cannot be greater than max_value"}), 400
```

  - This is a basic **comparison operation** with **O(1)** complexity.

4.5. Data Preloading

CSV data is preloaded into a **Pandas DataFrame** during app startup:

```
data_df = pd.read_csv(file_path, encoding='utf-8-sig')
```

- **Time Complexity**: Loading a CSV file involves parsing and reading, which scales with the file size:
  - **O(n * m)** where `n` is the number of rows and `m` is the number of columns.

4.6. Concepts Relevant to DSA

- **Greedy Filtering**: The filtering mechanism can be seen as a greedy algorithm, where each condition narrows down the dataset without backtracking.
- **Data Preprocessing**: Loading the entire CSV into memory (a DataFrame) is akin to preprocessing data for efficient querying later, reducing overhead during requests.

- **Key-Value**: The use of dictionaries for `search_conditions` is a practical example of hash-based storage for fast lookups.
- **Sequential Access**: Operations like string matching and numeric range filtering iterate sequentially over the dataset (conceptually similar to traversing a linked list or array).

5. SET UP ENVIRONMENT

Docker is a platform that allows developers to easily build, deploy, and run applications using containers. Containers are lightweight, portable, and self-sufficient, containing everything needed to run a piece of software: source code, runtime, libraries, and dependencies. Docker provides a way to package and distribute applications in these isolated environments (containers), ensuring that they run consistently across different machines or systems.

In the project folder, open the terminal:

- Build the *Docker image*: navigate to the directory containing the Dockerfile and ***docker-compose.yml***, then run: **docker-compose build**
- Run the application: once the image is built, start the application: **docker-compose up**
- This will start the **Flask** application on port 5000. Open a web browser or use curl to access the application at **http://localhost:5000**.
- To finish: press CTRL+C in the terminal or use: **docker-compose down**.

6. REFERENCES

W3Schools. (n.d.) *Pandas Tutorial*. Retrieved December 2024, from: https://www.w3schools.com/python/pandas/.

GeeksforGeeks. (n.d.) *Pandas Read CSV in Python*. Retrieved December 2024, from: https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/.

GeeksforGeeks. (n.d.) *Pandas DataFrame*. Retrieved December 2024, from: https://www.geeksforgeeks.org/python-pandas-dataframe/.