Cash Flow Minimizer Using Data Structures

Abstract:

This project presents a **Cash Flow Minimizer** system aimed at optimizing financial transactions within a group or organization. It leverages efficient **data structures and algorithms** to reduce the number of cash transactions required to settle debts among participants.

Key data structures include **Graphs** to model transactions, **Heaps** and **Priority Queues** to prioritize settlements, and **Stacks/Queues** for transaction history and undo features. The process minimizes the total number of payments while maintaining accuracy and speed.

This project showcases the real-world application of data structures in financial systems, emphasizing optimization, algorithmic thinking, and efficient problem-solving.

Introduction:

Managing and minimizing cash flows in group transactions (e.g., friends splitting bills, business expense sharing) is a classic optimization problem. Rather than each individual paying others directly, a minimized set of transactions can significantly reduce complexity and transaction costs.

Data Structures and Algorithms (DSA) play a crucial role in solving this problem. A **graph** structure models debt relationships, **priority queues** manage the largest debts and credits, and **hash maps** allow quick access to participant balances. These structures work together to simplify cash settlements. Beyond personal finance, this system can be extended to **corporate accounting**, **blockchain applications**, and **decentralized finance (DeFi)** where transactional efficiency is key.

Problem Statement

Designing a system to minimize the number of cash transactions requires addressing multiple challenges:

1. Optimal Transaction Reduction

- Eliminate unnecessary transactions by offsetting debts.
- Minimize the number of payments to achieve balance.

2. Efficient Balance Calculation

- Compute net balances for all participants after multiple transactions.
- Handle positive (creditor) and negative (debtor) balances.

3. Interactive and Scalable System

- Support undo/redo for transactions.
- Provide real-time optimization suggestions.
- Adapt to growing group sizes or transaction volumes.

Data Structures Used

1. Graph Representation for Debts

- *Why Graph?*
  Graphs allow modeling transactions as directed edges (A owes B) with weights (amount).
- *Role:*
  - Represents all debts within a group.
  - Helps identify cycles that can be eliminated.

2. Hash Map for Net Balances

- *Why Hash Map?*
  Quick access to each participant's total credit or debt.
- *Role:*
  - Stores net balances (sum of owed and lent amounts).
  - Helps separate creditors and debtors efficiently.

3. Min-Heap and Max-Heap (Priority Queue)

- *Why Heaps?*
  Efficient retrieval of the largest debtor and creditor at each step.
- *Role:*
  - Select top participants to settle transactions.
  - Balance debts in logarithmic time complexity.

4. Stack for Undo/Redo

- *Why Stack?*
  LIFO structure suits reverting recent transactions.
- *Role:*
  - Record each transaction for undo.
  - Enable redo by storing reversed actions in another stack.

5. Queue for Transaction History

- *Why Queue?*
  FIFO structure for tracking the order of transactions.
- *Role:*
  - Display history chronologically.
  - Support reporting and analysis.

Algorithm Explanation:

Step-by-Step Breakdown:

1. *Input Transaction Data*
    - Accept input in the form of "Person A pays Person B $X."
    - Update balances in a hash map.

2. *Calculate Net Balances*
    - Sum incoming and outgoing payments.
    - Separate participants into creditors and debtors.

3. *Minimize Transactions Using Heaps*
    - Use a max-heap for creditors and a min-heap for debtors.
    - Pop top creditor and debtor, settle minimum of the two.
    - Update balances and reinsert into heap if non-zero.

4. *Use Stack for Undo/Redo*

- Push each settlement onto a stack.
- Allow reversal and re-execution of transactions.

5. *Display Results*
    - Show minimized list of transactions.
    - Include graphical representation or table format.

Implementation Overview:

1. Balance Calculator (Hash Map)

- Tracks total in/out per participant.
- O(1) update and lookup.

2. Minimize Transactions (Heap + Greedy Algorithm)

- Match highest debtor with highest creditor.
- Settle in minimal steps.

3. Transaction History (Queue)

- Logs all settlements in order.
- Useful for reports or debugging.

4. Undo/Redo Feature (Stack)

- Allows reversal of recent settlements.

Execution Screenshots:

```
Welcome to the Cash Flow Minimizer!
How many people are involved in transactions? 4

Enter who owes whom (Person i to Person j):
Amount Person 1 owes to Person 2: 30
Amount Person 1 owes to Person 3: 40
Amount Person 1 owes to Person 4: 10
Amount Person 2 owes to Person 1: 20
Amount Person 2 owes to Person 3: 30
Amount Person 2 owes to Person 4: 0
Amount Person 3 owes to Person 1: 40
Amount Person 3 owes to Person 2: 50
Amount Person 3 owes to Person 4: 10
Amount Person 4 owes to Person 1: 0
Amount Person 4 owes to Person 2: 0
Amount Person 4 owes to Person 3: 20

Calculating simplified payments...

=> Person 1 pays Rs. 20 to Person 2
=> Person 3 pays Rs. 10 to Person 2

All debts settled with minimum number of transactions.



=== Code Execution Successful ===
```

```
Welcome to the Cash Flow Minimizer!
How many people are involved in transactions? 2

Enter who owes whom (Person i to Person j):
Amount Person 1 owes to Person 2: 1000
Amount Person 2 owes to Person 1: 500

Calculating simplified payments...

=> Person 1 pays Rs. 500 to Person 2

All debts settled with minimum number of transactions.


=== Code Execution Successful ===
```

Github link:

[shivam-9188/Minor-project]

REFRENCE LINK –

[GeeksforGeeks | A computer science portal for geeks]

Conclusion:

This project successfully implements a **cash flow minimizer** that reduces the number of financial transactions among a group using efficient data structures. Through **Graphs**, **Heaps**, **Hash Maps**, **Stacks**, and **Queues**, the system optimizes settlement operations and enhances financial clarity.

By applying core principles of DSA, this project highlights the power of algorithmic optimization in solving everyday problems, making it a valuable asset for fintech developers, accountants, and data science learners.