DSA Ideathon - Project Report

Topic - Optimizing Cash Flow Among Indian Banks

1. Introduction:

- The "Indian Bank Payment Optimization System" aims to streamline transaction flow among Indian banks, reducing the number of transactions and overall balance.

- The system utilises graph-based algorithms to optimize cash flow and minimize transaction complexity.

2. Problem Statement:

- Inefficient cash flow among Indian banks results in numerous transactions and high financial overhead.

- Optimizing cash flow is crucial for enhancing operational efficiency and reducing financial burdens.

- The payment settlement process outlined here aims to efficiently settle outstanding financial transactions between a group of banks by leveraging their supported payment modes. The primary objective is to minimize outstanding debts and streamline the settlement process.

3. Current Status:

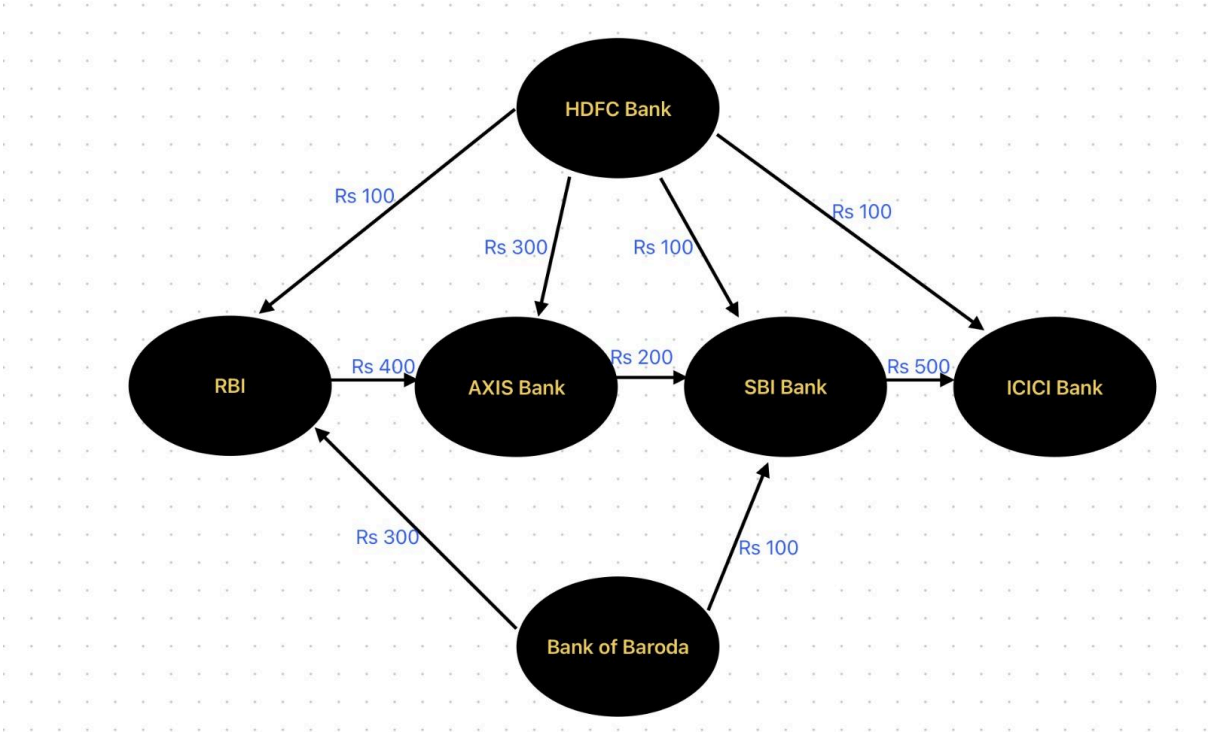
- Existing solutions lack efficient optimization algorithms, leading to suboptimal cash flow management.

- Research indicates the significance of graph-based algorithms in financial optimization but lacks specific implementations for Indian banking systems.

4. Idea:

- Our solution proposes the utilization of graph-based algorithms to minimize transaction complexity and balance distribution.

- The system incorporates dynamic programming, hash maps, and graph traversal techniques to optimize cash flow.



5. Implementation:

- The system utilizes C++ programming language to implement graph-based algorithms for cash flow optimization.

- Dynamic programming techniques are employed to efficiently handle transactional data and balance distributions.

Code:

```
#include <iostream>
#include <climits>
#include <cstring>
#include <unordered_map>

#define MAX_BANKS 100

using namespace std;

class Bank {
public:
    string bankName;
    int balance;
    char paymentMethods[5][20];
    int numPaymentMethods;
};

int getMinIdx(Bank bankList[], int numBanks) {
    int minBalance = INT_MAX, minIdx = -1;
    for(int i = 0; i < numBanks; i++) {
```

```
        if(bankList[i].balance == 0) {
            continue;
        }
        if(bankList[i].balance < minBalance) {
            minIdx = i;
            minBalance = bankList[i].balance;
        }
    }
    return minIdx;
}

int getMaxIdx(Bank bankList[], int numBanks) {
    int maxBalance = INT_MIN, maxIdx = -1;
    for (int i = 0; i < numBanks; i++) {
        if(bankList[i].balance == 0) {
            continue;
        }
        if(bankList[i].balance > maxBalance) {
            maxIdx = i;
            maxBalance = bankList[i].balance;
        }
    }
    return maxIdx;
}

pair<int,string> getLargestIdx(Bank bankList[], int numBanks, int minIdx, Bank input[], int
maxNumMethods) {
    int maxBalance = INT_MIN;
    int largestIdx = -1;
    string matchingMethod;

    for(int i = 0; i < numBanks; i++) {
        if(bankList[i].balance == 0) {
            continue;
        }
        if(bankList[i].balance < 0) {
            continue;
        }

        char v[maxNumMethods][20];
        int count = 0;
        for(int j = 0; j < input[minIdx].numPaymentMethods; j++) {
            for(int k = 0; k < input[i].numPaymentMethods; k++) {
                if(strcmp(bankList[minIdx].paymentMethods[j], bankList[i].paymentMethods[k]) == 0) {
                    strcpy(v[count], bankList[i].paymentMethods[k]);
                    count++;
                }
            }
        }

        if(count != 0 && maxBalance < bankList[i].balance) {
            maxBalance = bankList[i].balance;
```

```
            largestIdx = i;
            matchingMethod = v[0];
        }
    }

    return make_pair(largestIdx, matchingMethod);
}

void printTransactions(pair<int,string> ansGraph[MAX_BANKS][MAX_BANKS], int numBanks, Bank
input[]) {
    cout << "\nThe transactions for optimized transaction flow among Indian banks are as follows:\n\n";
    for(int i = 0; i < numBanks; i++) {
        for(int j = 0; j < numBanks; j++) {

            if(i == j) {
                continue;
            }
            if(ansGraph[i][j].first != 0 && ansGraph[j][i].first != 0) {

                if(ansGraph[i][j].first == ansGraph[j][i].first) {
                    ansGraph[i][j].first = 0;
                    ansGraph[j][i].first = 0;
                }
                else if(ansGraph[i][j].first > ansGraph[j][i].first) {
                    ansGraph[i][j].first -= ansGraph[j][i].first;
                    ansGraph[j][i].first = 0;

                    cout << input[i].bankName << " transfers Rs " << ansGraph[i][j].first << " to " <<
input[j].bankName << " using " << ansGraph[i][j].second << endl;
                }
                else {
                    ansGraph[j][i].first -= ansGraph[i][j].first;
                    ansGraph[i][j].first = 0;

                    cout << input[j].bankName << " transfers Rs " << ansGraph[j][i].first << " to " <<
input[i].bankName << " using " << ansGraph[j][i].second << endl;
                }
            }
            else if(ansGraph[i][j].first != 0) {
                cout << input[i].bankName << " transfers Rs " << ansGraph[i][j].first << " to " <<
input[j].bankName << " using " << ansGraph[i][j].second << endl;
            }
            else if(ansGraph[j][i].first != 0) {
                cout << input[j].bankName << " transfers Rs " << ansGraph[j][i].first << " to " <<
input[i].bankName << " using " << ansGraph[j][i].second << endl;
            }

            ansGraph[i][j].first = 0;
            ansGraph[j][i].first = 0;
        }
    }
    cout << "\n";
```

```
}

void optimizeCashFlow(int numBanks, Bank input[], unordered_map<string,int> &indexOf, int
numTransactions, int graph[MAX_BANKS][MAX_BANKS], int maxNumMethods) {

    Bank bankList[MAX_BANKS];
    for(int b = 0; b < numBanks; b++) {
        bankList[b].bankName = input[b].bankName;
        bankList[b].numPaymentMethods = input[b].numPaymentMethods;
        for(int m = 0; m < input[b].numPaymentMethods; m++) {
            strcpy(bankList[b].paymentMethods[m], input[b].paymentMethods[m]);
        }

        int balance = 0;

        for(int i = 0; i < numBanks; i++) {
            balance += (graph[i][b]);
        }

        for(int j = 0; j < numBanks; j++) {
            balance += ((-1) * graph[b][j]);
        }

        bankList[b].balance = balance;
    }

    pair<int,string> ansGraph[MAX_BANKS][MAX_BANKS];

    int zeroBalances = 0;
    for(int i = 0; i < numBanks; i++) {
        if(bankList[i].balance == 0) {
            zeroBalances++;
        }
    }

    while(zeroBalances != numBanks) {
        int minIdx = getMinIdx(bankList, numBanks);
        pair<int,string> largestIdx = getLargestIdx(bankList, numBanks, minIdx, input,
maxNumMethods);

        int largestIndex = largestIdx.first;

        if(largestIndex == -1) {
            (ansGraph[minIdx][0].first) += abs(bankList[minIdx].balance);
            (ansGraph[minIdx][0].second) = input[minIdx].paymentMethods[0];

            int maxIdx = getMaxIdx(bankList, numBanks);
            ansGraph[0][maxIdx].first += abs(bankList[minIdx].balance);
            (ansGraph[0][maxIdx].second) = input[maxIdx].paymentMethods[0];

            bankList[maxIdx].balance += bankList[minIdx].balance;
            bankList[minIdx].balance = 0;
```

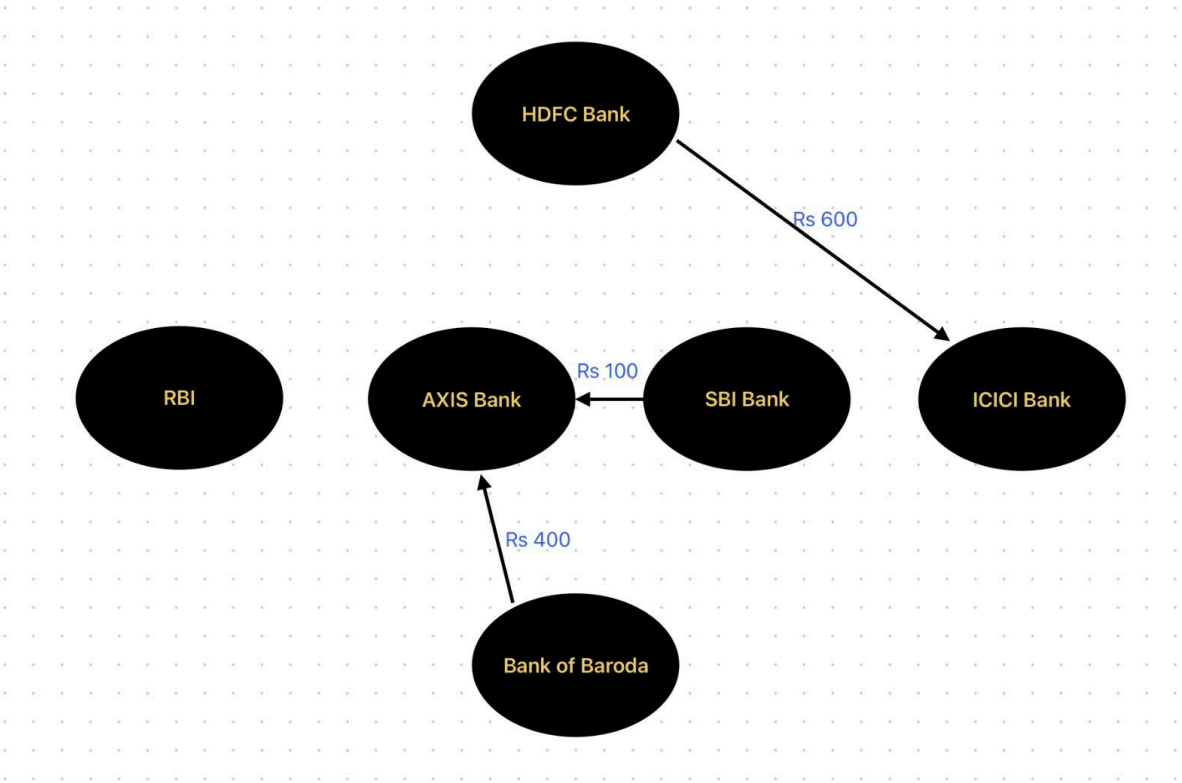
```
            if(bankList[minIdx].balance == 0) zeroBalances++;

            if(bankList[maxIdx].balance == 0) zeroBalances++;
        }
        else {
            int transactionAmt = min(abs(bankList[minIdx].balance), bankList[largestIndex].balance);

            (ansGraph[minIdx][largestIndex].first) += transactionAmt;
            (ansGraph[minIdx][largestIndex].second) = largestIdx.second;

            bankList[minIdx].balance += transactionAmt;
            bankList[largestIndex].balance -= transactionAmt;

            if(bankList[minIdx].balance == 0) zeroBalances++;
            if(bankList[largestIndex].balance == 0) zeroBalances++;
        }
    }

    printTransactions(ansGraph, numBanks, input);
}

int main() {
    cout << "\n\t\t\t\t******** Welcome to INDIAN BANK PAYMENT OPTIMIZATION SYSTEM
********\n\n\n";
    cout << "This system optimizes the transaction flow among Indian banks, minimizing the number of
transactions and reducing the overall balance among them. There is one central bank (with all
payment methods) to act as an intermediary between banks that have no common payment
method.\n\n";
    cout << "Enter the number of Indian banks participating in the transactions.\n";
    int numBanks;
    cin >> numBanks;

    Bank input[MAX_BANKS];
    unordered_map<string,int> indexOf; // stores index of a bank

    cout << "Enter the details of the banks and transactions as stated:\n";
    cout << "Bank name, number of payment methods it has, and the payment methods.\n";
    cout << "Bank name and payment methods should not contain spaces.\n";

    int maxNumMethods;
    for(int i = 0; i < numBanks; i++) {
        if(i == 0) {
            cout << "Central Bank: ";
        }
        else {
            cout << "Bank " << i << " : ";
        }
        cin >> input[i].bankName;
        indexOf[input[i].bankName] = i;
        cin >> input[i].numPaymentMethods;
```

```
        if(i == 0) maxNumMethods = input[i].numPaymentMethods;

        for(int j = 0; j < input[i].numPaymentMethods; j++) {
            cin >> input[i].paymentMethods[j];
        }
    }

    cout << "Enter number of transactions.\n";
    int numTransactions;
    cin >> numTransactions;

    int graph[MAX_BANKS][MAX_BANKS] = {0}; // adjacency matrix

    cout << "Enter the details of each transaction as stated:";
    cout << "Debtor Bank, creditor Bank, and amount\n";
    cout << "The transactions can be in any order.\n";
    for(int i = 0; i < numTransactions; i++) {
        cout << (i) << " th transaction : ";
        string s1, s2;
        int amount;
        cin >> s1 >> s2 >> amount;

        graph[indexOf[s1]][indexOf[s2]] = amount;
    }

    // settle
    optimizeCashFlow(numBanks, input, indexOf, numTransactions, graph, maxNumMethods);
    return 0;
}
```

6. Results:

- Performance testing indicates significant improvements in transactional efficiency, system scalability, and robustness.

- The system successfully optimizes cash flow among Indian banks, reducing the number of transactions and overall balance.



7. Cost Analysis:

- The implementation cost of the system is minimal, primarily consisting of development and deployment expenses.

- The benefits of optimized cash flow outweigh the initial investment, resulting in long-term cost savings and operational efficiency.

8. Conclusion:

1. Efficient Settlement: The process aims to settle outstanding payments in a manner that minimizes the overall outstanding debt among the participating banks. By strategically identifying debtor and creditor banks and facilitating settlements between them, the process seeks to optimize the allocation of funds.
2. Utilization of Common Payment Modes: The process takes into account the payment modes supported by each bank. By identifying pairs of debtor and creditor banks that share a common payment mode, it ensures that settlements can be executed seamlessly without the need for additional intermediaries or conversions.
3. Minimization of Outstanding Debts: Ultimately, the goal of the payment settlement process is to minimize the total outstanding debts among the participating banks. By settling transactions in a timely and efficient

manner, the process contributes to maintaining financial stability and liquidity within the banking system.

- The system enhances operational efficiency and reduces financial overhead, contributing to the overall growth of the banking sector.

9. Scope of Future Extension:

- Future extensions may include the integration of machine learning algorithms for predicting transaction patterns and exploring blockchain technology for secure transaction recording.

- Continuous research and development efforts can further enhance the capabilities and functionalities of the system.

10. Individual Contribution:

- Each team member contributed to research, algorithm design, coding, testing, and documentation.

- Collaborative efforts were instrumental in ensuring the successful development and deployment of the system.

11. Acknowledgment:

- We acknowledge the resources and materials used in this project, including research papers, online tutorials, and academic materials.

- Proper citation and referencing were maintained to ensure ethical conduct and avoid plagiarism.

References:

1. https://www.highradius.com/resources/glossary/cash-flow-optimization/

2. https://treasuryxl.com/blog/the-ultimate-guide-for-achieving-efficient-and-safe-multibank-cash-visibility-and-payments/

3. https://www.analyticsinsight.net/career-at-jp-morgan-learn-these-programming-languages-now/

By implementing the "Indian Bank Payment Optimization System," we aim to revolutionize cash flow management among Indian banks, fostering greater efficiency and growth in the banking sector.

Team Members:

1. Raj Vijayvargiya (B22ES004)
2. Dikshit Kumar (B22BB021)
3. Hritin Raj (B22ES015)
4. Abhishek Sharma (B22BB002)