CASH FLOW MINIMIZAL

A MINI PROJECT REPORT

18CSC204J- DESIGN ANALYSIS OF ALGORITHMS LABORATORY

Submitted by

HARSHITHA GOKARAJU (RA2011030010020)

RAKSHITA RAJ (RA2011030010020)

Under the guidance of

Dr.B Yamini

(Assistant Professor, Department of Networking and Communications)

In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING with specialization in CYBER SECURITY

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603203

JUNE 2022

KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this mini project report titled "**CASH FLOW MINIMIZAL"** is the bonafide work of "HARSHITHA GOKARAJU (RA2011030010020), RAKSHITA RAJ (RA2011030010030) who carried out the mini project work and the Laboratory exercises under my supervision for 18CSC204J- Design and Analysis of Algorithms Laboratory. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Dr.B Yamini
ASSISTANT PROFESSOR
18CSC204J- Design and Analysis of Algorithms
Course Faculty
Department of Networking and Communications

Table of content:

Contribution Table:

HARSHITHA DEVI RA2011030010020	Searching for the problem statement and suggested the Design technique used
RAKSHITA RAJ RA2011030010030	Code for the problem according to the algorithm and Complexity analysis

Problem Definition:

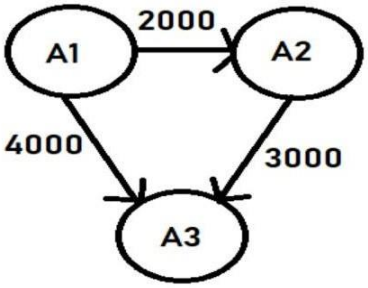
This project aims in implementing a software system which regulates a company's payments and transactions effectively and conveniently. They streamline the transactions in a non-loss manner**.**

Problem Explanation with diagram and example:

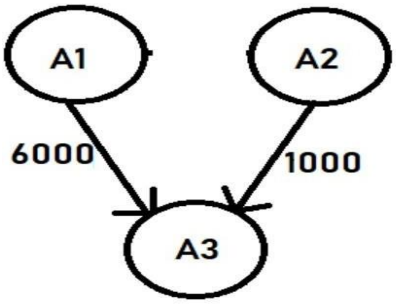
The CASH FLOW MINIMIZAL project is a software which focuses on reducing the extra cash lost during transactions in a company's payments. It brings in a system which chooses an ideal path for the transaction and reduces the extra long processed transactions to simple direct transactions.
It is necessary because-

- An efficient transaction methodology which reduces complexity
- Reduces the amount of transactions done
- Making the services more efficient when dealing with a multi node high traffic client

- Time- efficient



A1 has to pay 2000 to A2 and 4000 to A3, A2 has to pay 3000 to A3



we can minimize the flow between A1 and A2 by directly paying to A3.

Design Techniques used:

The program makes use of the greedy algorithm. Any algorithm that follows the problem-solving technique of choosing the locally best option at each stage is known as a greedy algorithm. A greedy strategy may not produce an optimal solution in many cases, but it can produce locally optimal solutions that approximate a globally optimal solution in an acceptable amount of time. For example, a greedy strategy for the travelling salesman problem is to "visit the nearest unvisited city at each step of the route." This technique does not aim to discover the best solution, but it does it in a fair number of steps; finding the best solution to such a difficult problem usually takes an unreasonable amount of time.

Greedy methods are used in mathematical optimization to solve combinatorial problems with matroid qualities and to yield constant-factor approximations to optimization problems with the submodular structure.

Algorithm for the problem:

- The project uses GREEDY ALGORITHM.

Explanation of algorithm with example:

The greedy approach is used to build the solution in pieces, and this is what we want to minimize the cash flow. At every step, we will settle all the amounts of one person and recur for the remaining n-1 persons.

How to pick the first person? To pick the first person, calculate the net amount for every person where net amount is obtained by subtracting all debts (amounts to pay) from all credits (amounts to be paid). Once net amount for every person is evaluated, find two persons with maximum and minimum net amounts. These two persons are the most creditors and debtors. The person with minimum of two is our first person to be settled and removed from list. Let the minimum of two amounts be x. We pay 'x' amount from the maximum debtor to maximum creditor and settle one person. If x is equal to the maximum debit, then maximum debtor is settled, else maximum creditor is settled.

The basic objective of cash flow management is to understand and minimize the cost of processing and managing cash flows both externally with other companies and organizations and internally within the company or group.

Significant cost savings and improvements in efficiency can be achieved in Cash Flow Management through the use of cash flow forecasting, in-house banking, netting and payment factories.

Forecasting of Cash Flows is one of the most complex and difficult tasks in cash and treasury management. It requires commitment of the whole company to achieve cost-effective and accurate forecasts in the four types of cash flow.

In-house Banking is an internal treasury function, typically used by large multi-national corporations, where the treasury department provides foreign exchange, intra-group lending and cash flow management services for the group.

Multi-lateral Netting systems and services minimize the number of payments and the cost of FX between subsidiaries and companies within a group. Also for some external payments by making one payment, at the end of a given period, for the net cash flow between either intra-group or with trading partners.

Payment Factories minimize the cost of external payment flows as well as the internal cost of processing payments by centralizing all payments in a dedicated processing centre. Payment factories can also be used to optimize liquidity by managing the timing of payments. Some well established payment factories also provide payment collection services.

You are given a list of ‘transactions’ between ‘n’ number of friends. who have to give each other money. The list consists of data of receiver, sender, and transaction.

Your task is to minimize the cash flow and the total number of transactions should also be minimum.

1. The implementation starts by Calculating the net amount for every person, which can be calculated by subtracting all the amount to be paid to him.
2. Net amount = sum(received money) - sum(sent money).
3. Find the person with maximum credit and maximum debit.
4. Assign them a variable as credit for the person with maximum credit and debit for the person with maximum debit.
5. Respectively assign a variable as max_cred for maximum amount to be credited and max_debt for maximum amount to be debited.
6. Find the minimum of the two and name them as y.
7. If y equals max_cred, delete cred from the list and repeat for the remaining (n-1) people.
8. If y equals max_deb, delete debt from the group of people and repeat for recursion.
9. If the amount is 0, then the settlement is done

Code:

```
class Solution
{
//Total no of
persons  static
final int n = 3;

//Returns the index of minimum value in
arr[] static int get_min(int arr[])
{ int in = 0;
for (int i = 1;
i < n; i++) if
(arr[i] <
arr[in]) in = i;
return in;
}

//Returns the maximum index in
arr[] static int get_max(int
arr[])
{ int in = 0;
for (int i = 1;
i < n; i++) if
(arr[i] >
arr[in]) in = i;
return in;
}

//Function which return the minimum of 2
values static int min_two(int x, int y)
```

```
{ return (x
< y) ? x:
y;
}



//Here amount array is storing the net amount to be
settled
//to/from person p(i), now if amount[p] is +ve then
ith person
//give amount[i] otherwise amount[p] will give -
amount[i].
 static void
min_cashRec(int amount[])
{
// Find the indexes of minimum and
// maximum values in amount[]
// amount[max_credit] indicates the
maximum amount // that to be given to the
person.
// And amount[max_debit] indicates the maximum
amount
// to be taken from a person.
//Along with the positive value there will also be
negative value int max_credit = get_max(amount),
max_debit = get_min(amount);


//amounts to be 0 for the settlement if
(amount[max_credit] == 0 &&
amount[max_debit] == 0) return;
```

```
// Find the minimum of two amounts int min =
min_two(-amount[max_debit],
amount[max_credit]); amount[max_credit] -= min;
amount[max_debit] += min;

// If the minimum is the maximum amount to be
System.out.println("Person " + max_debit + " pays "
+ min
+ " to " + "Person " + max_credit);

//recur for remaining persons
min_cashRec(amount);
}

// Given a set of persons as graph[]
// where graph[i][j] indicates
// the amount that person i needs to
// pay person j, this
function //finds and
settles the debts.
static void min_cash(int graph[][])
{
// Create an array amount[],
// initialize all value in it as 0 for storing
//the net amount.
int amount[]=new int[n];

// Calculate the net amount to
// be paid to person 's', and
// stores it in amount[s]. The
// value of amount[s] an be
```

```
//calculated by subtracting
//sum of all received money - sum of all sent
money.
 for (int s = 0; s < n; s++)
for (int i = 0; i < n; i++)
amount[s] += (graph[i][s] -
graph[s][i]);

min_cashRe
c(amount);
}

// Main function
public static void main (String[] args)
{
// cash[i][j] means that
the amount //person i has
to pay to person j. int
cash[][] = { {0, 2000,
4000},
{0, 0, 3000},
{0, 0, 0},};

// Print the solution
min_cash(cash);
}
}
```

```cpp
 #include <bits/stdc++.h>
using namespace std;
class DSU {
    int* parent;
    int* rank;
public:
    DSU(int n)
    {
        parent = new int[n];
        rank = new int[n];

        for (int i = 0; i < n; i++) {
            parent[i] = -1;
            rank[i] = 1;
        }
    }
// Find function
    int find(int i)
    {
        if (parent[i] == -1)
            return i;
return parent[i] = find(parent[i]);
    }
    // union function
    void unite(int x, int y)
```

```
    void unite(int x, int y)
    {
        int s1 = find(x);
        int s2 = find(y);

        if (s1 != s2) {
            if (rank[s1] < rank[s2]) {
                parent[s1] = s2;
                rank[s2] += rank[s1];
            }
            else {
                parent[s2] = s1;
                rank[s1] += rank[s2];
            }
        }
    }
};

class Graph {
    vector<vector<int> > edgelist;
    int V;

public:
    Graph(int V) { this->V = V; }

    void addEdge(int x, int y, int w)
```

```
void addEdge(int x, int y, int w)
{
    edgelist.push_back({ w, x, y });
}

void kruskals_mst()
{
    // 1. Sort all edges
    sort(edgelist.begin(), edgelist.end());

    // Initialize the DSU
    DSU s(V);
    int ans = 0;
    cout << "Following are the edges in the "
            "constructed MST"
         << endl;
    for (auto edge : edgelist) {
        int w = edge[0];
        int x = edge[1];
        int y = edge[2];

        // take that edge in MST if it does form a cycle
        if (s.find(x) != s.find(y)) {
            s.unite(x, y);
            ans += w;
```

```
                ans += w;
                cout << x << " -- " << y << " == " << w
                     << endl;
            }
        }
        cout << "Minimum Cost Spanning Tree: " << ans;
    }
};
int main()
{
    Graph g(12);
    g.addEdge(0, 1, 6);
    g.addEdge(1, 4, 2);
    g.addEdge(4,5,4);
    g.addEdge(5,7,10);
    g.addEdge(7,10,25);
    g.addEdge(10,11,3);
    g.addEdge(11,9,8);
    g.addEdge(9,3,18);
    g.addEdge(3,0,6);
    g.addEdge(4,2,7);
    g.addEdge(5,6,11);
    g.addEdge(7,6,22);
    g.addEdge(7,8,12);
```

```
    g.addEdge(7,6,22);
    g.addEdge(7,8,12);
    g.addEdge(8,9,1);
    g.addEdge(2,3,2);
    g.addEdge(2,6,2);
    g.addEdge(1,2,6);
    g.addEdge(6,8,2);
    g.addEdge(8,10,16);
    g.addEdge(1,2,1);
    g.kruskals_mst();
    return 0;
}
```

```
Following are the edges in the constructed MST
1 -- 2 == 1
8 -- 9 == 1
1 -- 4 == 2
2 -- 3 == 2
2 -- 6 == 2
6 -- 8 == 2
10 -- 11 == 3
4 -- 5 == 4
0 -- 1 == 6
11 -- 9 == 8
5 -- 7 == 10
Minimum Cost Spanning Tree: 41

...Program finished with exit code 0
Press ENTER to exit console.
```

Output:

Person0 pays 6000 to

person2 Person1 pays

1000 to person2

Time Complexity:

The time complexity of minimizing the cash flow is O(n^2).

n = number of persons

Conclusion:

Therefore, it was proved that cash flow minimizer problem can be solved using greedy method, with a better time complexity rather than using brute force technique.

References:

https://www.codingninjas.com/codestudio/problem-details/minimize-cash-flow-among-a-given-set-of-friends-who-have-borrowed-money- from-each-other_1170048other_1170048

https://www.geeksforgeeks.org/minimize-cash-flow-among-given-set-friends-borrowed-money/

https://www.tutorialspoint.com/minimize-cash-flow-among-a-given-set-of-friends-who-have-borrowed-money-from-each-other-in-cplusplus

https://github.com/soumyasethy/ShortestPath-CashFlow-Algorithm-Splitwise