Cash flow minimizer

PHANTOMS

22-01-2024

—

Data structures & Algorithms
(CS-401)

—

Mr. Shehzad

Contents

TEAM MEMBERS:

1. Maleeha	2022-uam-1925
2. Abdul Wahid	2022-uam-1910
3. Afia	2022-uam-1912
4. Fizza Sajjad	2022-uam-1911
5. Ali Usman	2022-uam-1960
6. Asad Ali	2022-uam-1931

INTRODUCTION

In this report, we present our semester project of cash flow minimizer, which is an application of data structures and algorithms to a real-world scenario. The cash flow minimizer is a program that aims to reduce the number of transactions among a group of people who have borrowed or lent money from each other. The program takes as input the amount of money that each person owes or is owed by others. The program outputs the optimal way to settle the debts using the minimum number of transactions. The program uses a **greedy algorithm** that iteratively finds the person with the minimum net amount and the person with the maximum net amount and a common payment mode, and settles the debt between them. The program also uses a **graph data structure** to store and update the transaction amounts between the people. The program is implemented in Java, python, C++, C#, C, R and tested on various input cases. The report describes the problem statement, the algorithm design, the data structure choice, the implementation details, the testing results, and the analysis of the program's performance and limitations.

ALGORITHM

An approach of greedy algorithm is used here.

1. Compute the net amount for every person. The net amount for person 'i' can be computed by subtracting sum of all debts from sum of all credits.
2. Find the two persons that are maximum creditor and maximum debtor. Let the maximum amount to be credited maximum creditor be maxCredit and maximum amount to be debited from maximum debtor be maxDebit. Let the maximum debtor be Pd and maximum creditor be Pc.

3. Find the minimum of maxDebit and maxCredit. Let minimum of two be x. Debit ‘x’ from Pd and credit this amount to Pc
4. If x is equal to maxCredit, then remove Pc from set of persons and recur for remaining (n-1) persons.
5. If x is equal to maxDebit, then remove Pd from set of persons and recur for remaining (n-1) persons.

Data structure:

The data structure we used is **graph** because of following reasons:

1. **Representation of Complex Relationship**
   Graphs are ideal for representing complex networks of relationships or interactions between entities, which in this case are the financial transactions between individuals. Each node represents an individual, and each edge represents a financial transaction (or debt) between two individuals.
2. **Modeling of direct relationships**
   In financial transactions, the direction of the transaction is crucial (who owes whom). Graphs naturally accommodate directional relationships through directed edges, making them suitable for modeling debts where direction (from debtor to creditor) is important.
3. **Simplification of Complex calculations**

   The minimization of cash flow involves complex calculations to reduce the number of transactions while settling all debts. Graph algorithms can efficiently handle these calculations. By representing debts as edges in a graph, the program can use algorithms to find the optimal way to minimize the total number of transactions.

EXPLANATION

Imagine a group of three friends: Alice, Bob, and Charlie. They owe each other money as follows:
Alice owes Bob $30.

Bob owes Charlie $20.
Charlie owes Alice $10.

Steps of the Program

Initialization:

The program starts by asking if the user wants to enter names for the persons.
It then prompts for the number of persons (in our case, 3 - Alice, Bob, Charlie).

Adding Nodes:

Nodes representing each person are added to the graph.
If names are provided, they are used; otherwise, default names are assigned (e.g., Person 1, Person 2).

Adding Edges:

The program asks for the amount each person owes to every other person.
For our scenario, the inputs would be:
Alice owes Bob $30.
Bob owes Charlie $20.
Charlie owes Alice $10.
These debts are represented as directed edges in the graph.

Calculating Net Amounts:

The program calculates the net amount each person owes or is owed.
In our scenario:
Alice's net amount: -$20 (owes $30, is owed $10)
Bob's net amount: +$10 (is owed $30, owes $20)
Charlie's net amount: +$10 (is owed $20, owes $10)

Minimizing Cash Flow:

The program finds the persons with the maximum amount to be received (maxCreditor) and the maximum amount to be paid (maxDebtor).
It then minimizes transactions by making the debtor pay the creditor directly.
For our scenario:
Alice pays $20 to Bob. (Alice's and Bob's net amount becomes $0)
The number of transactions is minimized from 3 to 1.

Output:

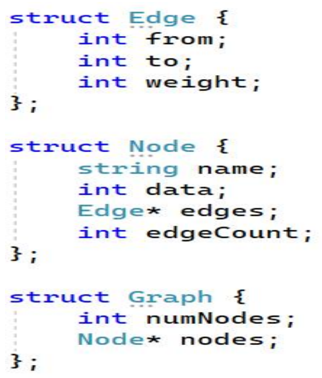
The program outputs the minimized transactions.
In this case, it will display: "Alice pays $20 to Bob."

IMPLEMENTATION

We will be using C++ to explain the implementation of program.

1. Initialize Nodes and Edges of Graph

First of all we have to initialize nodes and edges of our graph using struct where each node represents a person and each edge represents the weight or balance between each person



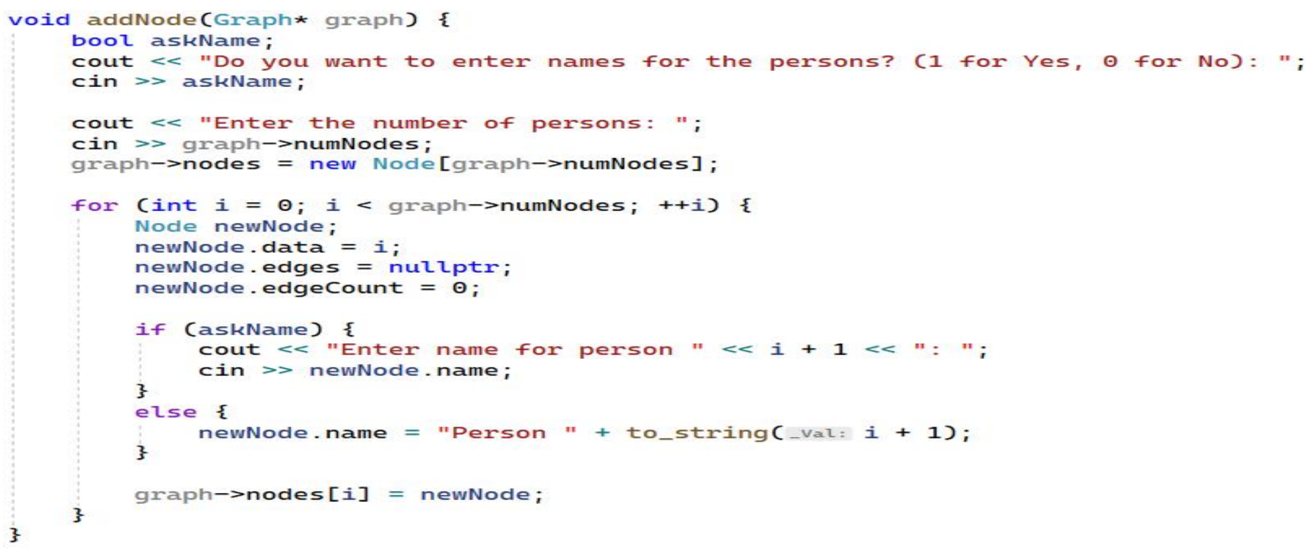
```
struct Edge {
    int from;
    int to;
    int weight;
};

struct Node {
    string name;
    int data;
    Edge* edges;
    int edgeCount;
};

struct Graph {
    int numNodes;
    Node* nodes;
};
```

It is shown above how we initialize our nodes and edges.

2. Function to add persons (Nodes)

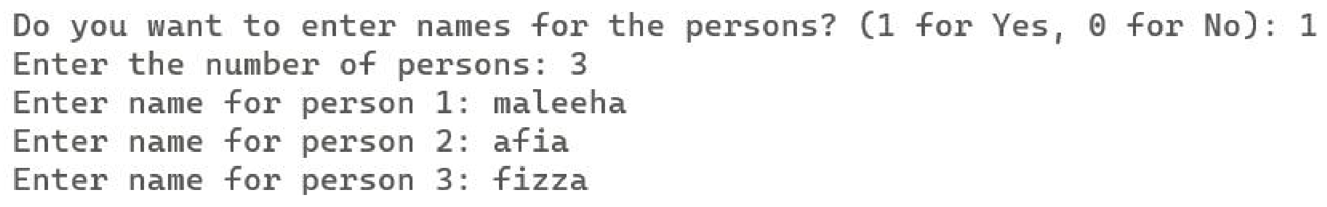
Now we divided this as:

- First ask user if he wants to add names of persons as it is easy but some people can skip this step as it can be time consuming
- Next user will be asked to enter number of persons
- Now nodes will be made according to data input by user.



```
void addNode(Graph* graph) {
    bool askName;
    cout << "Do you want to enter names for the persons? (1 for Yes, 0 for No): ";
    cin >> askName;

    cout << "Enter the number of persons: ";
    cin >> graph->numNodes;
    graph->nodes = new Node[graph->numNodes];

    for (int i = 0; i < graph->numNodes; ++i) {
        Node newNode;
        newNode.data = i;
        newNode.edges = nullptr;
        newNode.edgeCount = 0;

        if (askName) {
            cout << "Enter name for person " << i + 1 << ": ";
            cin >> newNode.name;
        }
        else {
            newNode.name = "Person " + to_string(_Val: i + 1);
        }

        graph->nodes[i] = newNode;
    }
}
```

Output will be:



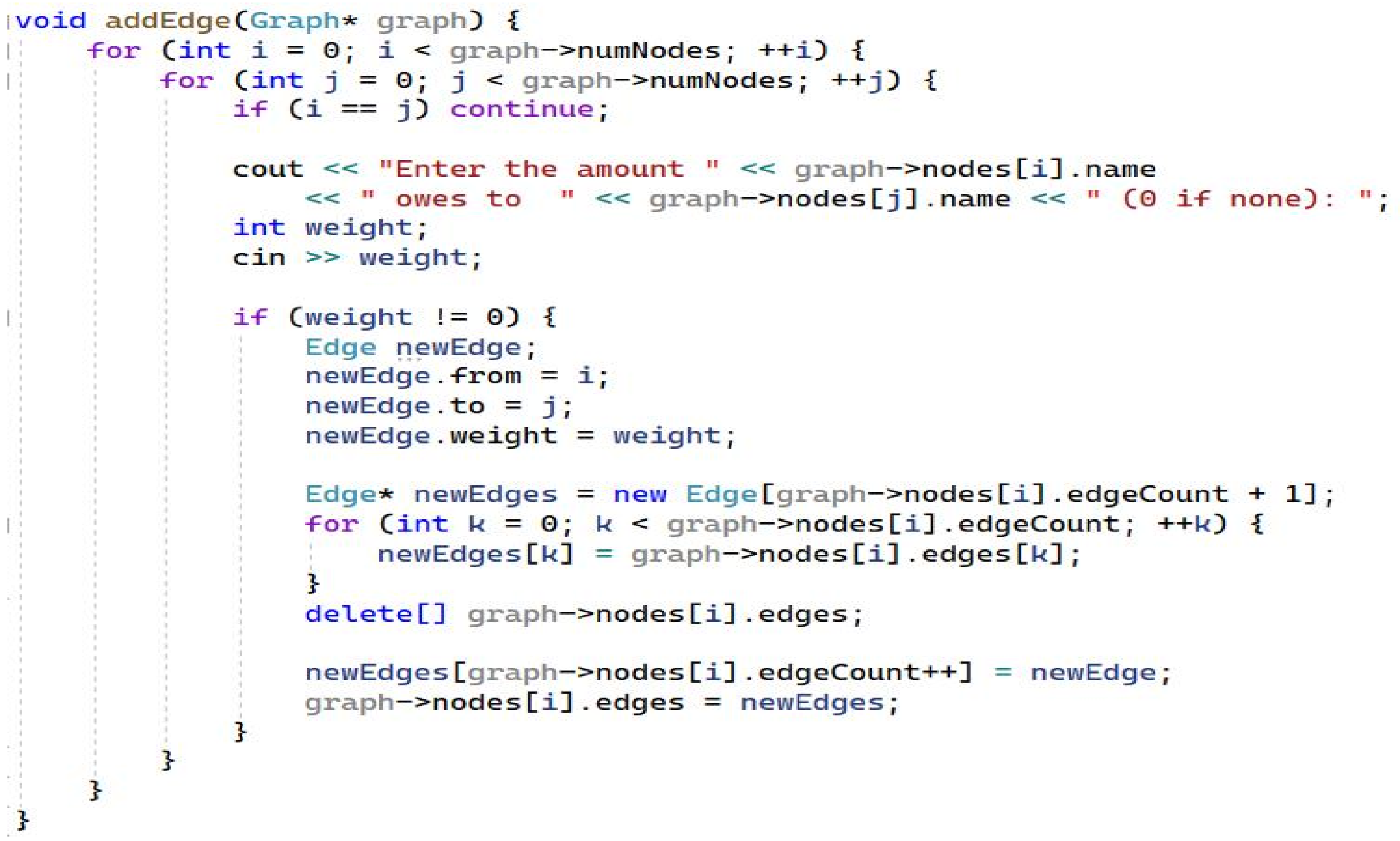
```
Do you want to enter names for the persons? (1 for Yes, 0 for No): 1
Enter the number of persons: 3
Enter name for person 1: maleeha
Enter name for person 2: afia
Enter name for person 3: fizza
```

So in this way the nodes will be added.

3. Add Edge (Balance)

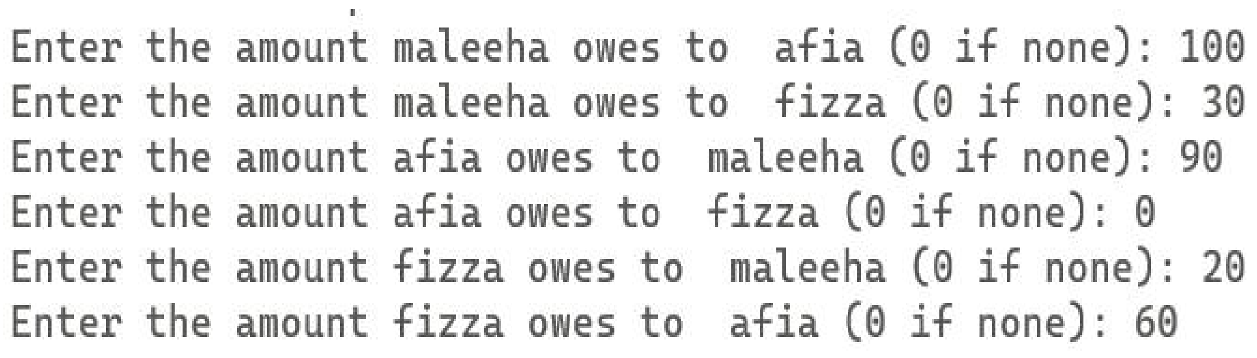
We divide this as:

- First we make two nested loops to make each possible transaction between two persons
- Then we ask user to enter person i owes to person j
- If balance input by user is not zero then edge is added to corresponding nodes



```
void addEdge(Graph* graph) {
    for (int i = 0; i < graph->numNodes; ++i) {
        for (int j = 0; j < graph->numNodes; ++j) {
            if (i == j) continue;

            cout << "Enter the amount " << graph->nodes[i].name
                << " owes to  " << graph->nodes[j].name << " (0 if none): ";
            int weight;
            cin >> weight;

            if (weight != 0) {
                Edge newEdge;
                newEdge.from = i;
                newEdge.to = j;
                newEdge.weight = weight;

                Edge* newEdges = new Edge[graph->nodes[i].edgeCount + 1];
                for (int k = 0; k < graph->nodes[i].edgeCount; ++k) {
                    newEdges[k] = graph->nodes[i].edges[k];
                }
                delete[] graph->nodes[i].edges;

                newEdges[graph->nodes[i].edgeCount++] = newEdge;
                graph->nodes[i].edges = newEdges;
            }
        }
    }
}
```

Output will be:

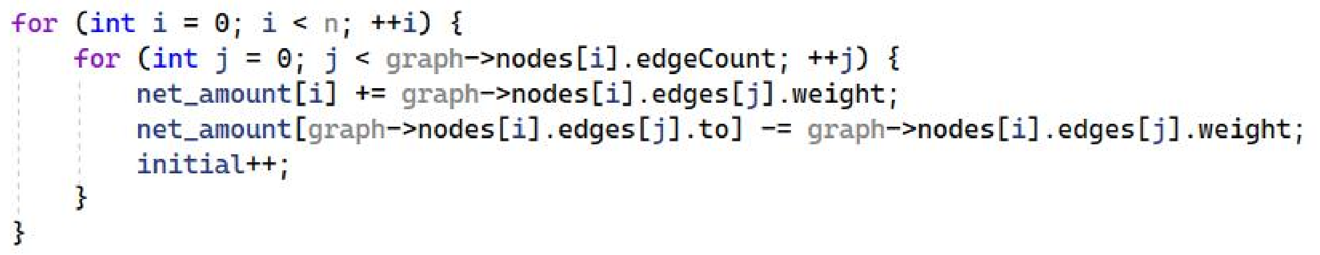


```
Enter the amount maleeha owes to  afia (0 if none): 100
Enter the amount maleeha owes to  fizza (0 if none): 30
Enter the amount afia owes to  maleeha (0 if none): 90
Enter the amount afia owes to  fizza (0 if none): 0
Enter the amount fizza owes to  maleeha (0 if none): 20
Enter the amount fizza owes to  afia (0 if none): 60
```

4. Minimizing the transactions

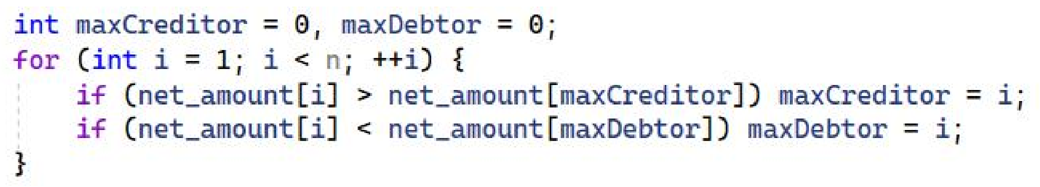
This can be divided as:

- First **settle the net amount** of each person by subtracting Sum of all the debts from sum of all the credits



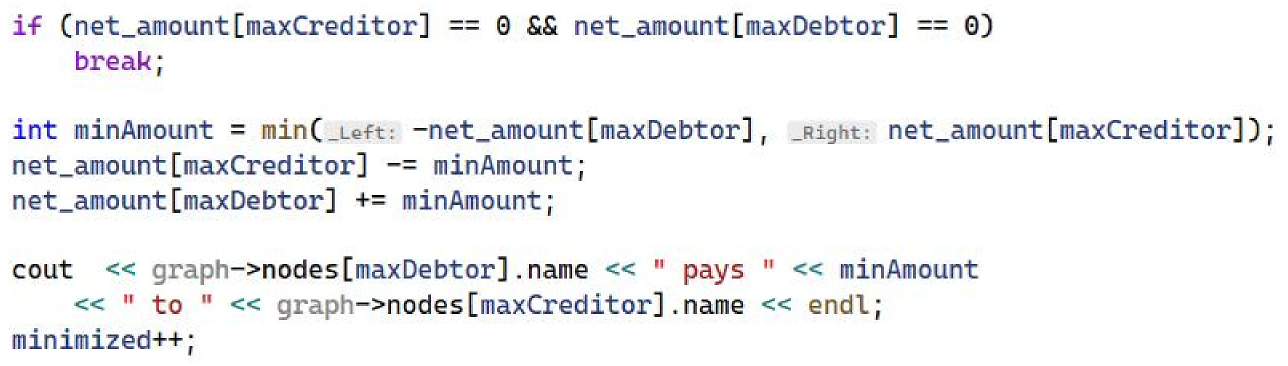
```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < graph->nodes[i].edgeCount; ++j) {
        net_amount[i] += graph->nodes[i].edges[j].weight;
        net_amount[graph->nodes[i].edges[j].to] -= graph->nodes[i].edges[j].weight;
        initial++;
    }
}
```

- After settling net amount **select maximum creditor and maximum debtor.**



```
int maxCreditor = 0, maxDebtor = 0;
for (int i = 1; i < n; ++i) {
    if (net_amount[i] > net_amount[maxCreditor]) maxCreditor = i;
    if (net_amount[i] < net_amount[maxDebtor]) maxDebtor = i;
}
```

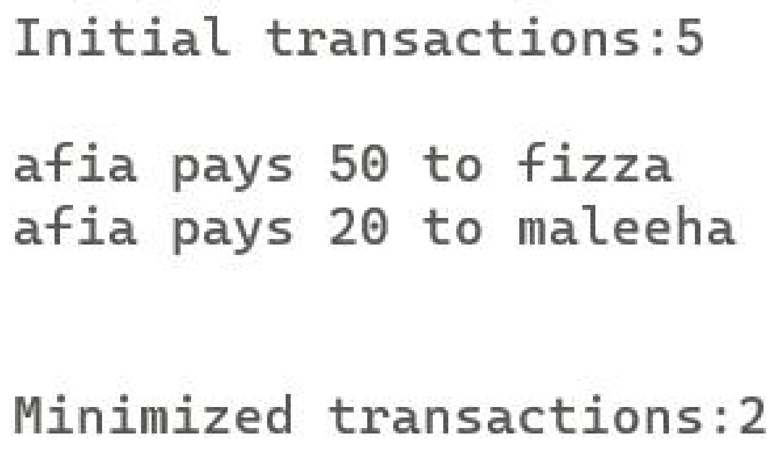
- After that settle the amount between these two



```
if (net_amount[maxCreditor] == 0 && net_amount[maxDebtor] == 0)
    break;

int minAmount = min(_Left: -net_amount[maxDebtor], _Right: net_amount[maxCreditor]);
net_amount[maxCreditor] -= minAmount;
net_amount[maxDebtor] += minAmount;

cout  << graph->nodes[maxDebtor].name << " pays " << minAmount
    << " to " << graph->nodes[maxCreditor].name << endl;
minimized++;
```

- In the end repeat these steps until we find the minimum number of transactions by using while loop or recursive call but we are using while loop here.

Output after minimizing the transactions will be:



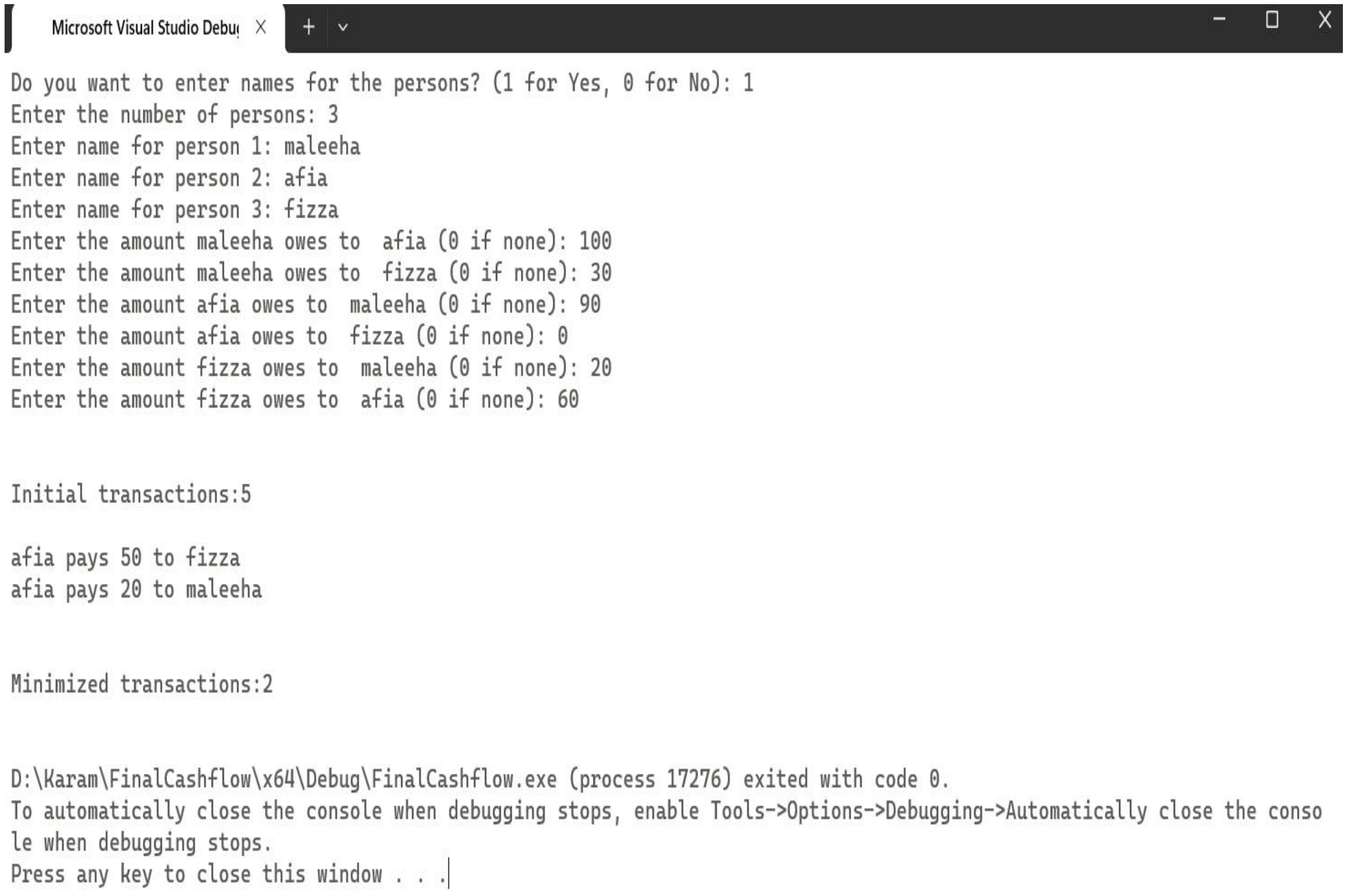
```
Initial transactions:5

afia pays 50 to fizza
afia pays 20 to maleeha


Minimized transactions:2
```

5. Main function

In the end we call the made functions in main program.

6. Output

Whole output will be



```
Do you want to enter names for the persons? (1 for Yes, 0 for No): 1
Enter the number of persons: 3
Enter name for person 1: maleeha
Enter name for person 2: afia
Enter name for person 3: fizza
Enter the amount maleeha owes to  afia (0 if none): 100
Enter the amount maleeha owes to  fizza (0 if none): 30
Enter the amount afia owes to  maleeha (0 if none): 90
Enter the amount afia owes to  fizza (0 if none): 0
Enter the amount fizza owes to  maleeha (0 if none): 20
Enter the amount fizza owes to  afia (0 if none): 60


Initial transactions:5

afia pays 50 to fizza
afia pays 20 to maleeha


Minimized transactions:2


D:\Karam\FinalCashflow\x64\Debug\FinalCashflow.exe (process 17276) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Code in R

```r
Edge <- function(from, to, weight) {
  structure(list(from = from, to = to, weight = weight), class = "edge")
}

Node <- function(name, data) {
  structure(list(name = name, data = data, edges = NULL, edgeCount = 0), class = "node")
}

Graph <- function(numNodes) {
  structure(list(numNodes = numNodes, nodes = vector("list", numNodes)), class = "graph")
}

addNode <- function(graph) {
  askName <- as.logical(readline("Do you want to enter names for the persons? (1 for Yes, 0 for
No): "))
  graph$numNodes <- as.integer(readline("Enter the number of persons: "))
  graph$nodes <- vector("list", graph$numNodes)

  for (i in 1:graph$numNodes) {
    newNode <- Node("", i - 1)
    newNode$edges <- NULL
    newNode$edgeCount <- 0

    if (askName) {
      newNode$name <- readline(paste("Enter name for person", i, ": "))
    } else {
      newNode$name <- paste("Person", i)
    }

    graph$nodes[[i]] <- newNode
  }
}

addEdge <- function(graph) {
  for (i in 1:graph$numNodes) {
    for (j in 1:graph$numNodes) {
      if (i == j) next

      weight <- as.integer(readline(paste("Enter the amount", graph$nodes[[i]]$name,
                                          "owes to", graph$nodes[[j]]$name, "(0 if none): ")))

      if (weight != 0) {
        newEdge <- Edge(i - 1, j - 1, weight)
        graph$nodes[[i]]$edges <- c(graph$nodes[[i]]$edges, newEdge)
        graph$nodes[[i]]$edgeCount <- graph$nodes[[i]]$edgeCount + 1
      }
    }
  }
}

min_cashflow <- function(graph) {
  n <- graph$numNodes
  net_amount <- rep(0, n)
  initial <- 0
  minimized <- 0

  for (i in 1:n) {
    for (j in 1:graph$nodes[[i]]$edgeCount) {
      net_amount[i] <- net_amount[i] + graph$nodes[[i]]$edges[[j]]$weight
      net_amount[graph$nodes[[i]]$edges[[j]]$to + 1] <- net_amount[graph$nodes[[i]]$edges[[j]]$to
+ 1] - graph$nodes[[i]]$edges[[j]]$weight
      initial <- initial + 1
    }
  }
  cat("\n\nInitial transactions:", initial, "\n\n")

  while (TRUE) {
    maxCreditor <- which.max(net_amount)
    maxDebtor <- which.min(net_amount)

    if (net_amount[maxCreditor] == 0 && net_amount[maxDebtor] == 0)
      break

    minAmount <- min(-net_amount[maxDebtor], net_amount[maxCreditor])
    net_amount[maxCreditor] <- net_amount[maxCreditor] - minAmount
    net_amount[maxDebtor] <- net_amount[maxDebtor] + minAmount

    cat(graph$nodes[[maxDebtor + 1]]$name, "pays", minAmount,
        "to", graph$nodes[[maxCreditor + 1]]$name, "\n")
    minimized <- minimized + 1
  }
  cat("\n\nMinimized transactions:", minimized, "\n\n")
}

graph <- Graph(0)
addNode(graph)
addEdge(graph)
min_cashflow(graph)
```

Code in Java

```java
import java.util.Scanner;

class Edge {
    int from;
    int to;
    int weight;
}

class Node {
    String name;
    int data;
    Edge[] edges;
    int edgeCount;
}

class Graph {
    int numNodes;
    Node[] nodes;
}

public class Main {
    static Scanner scanner = new Scanner(System.in);

    static void addNode(Graph graph) {
        boolean askName;
        System.out.print("Do you want to enter names for the persons? (1 for Yes, 0 for No): 
");
        askName = scanner.nextInt() == 1;

        System.out.print("Enter the number of persons: ");
        graph.numNodes = scanner.nextInt();
        graph.nodes = new Node[graph.numNodes];

        for (int i = 0; i < graph.numNodes; ++i) {
            Node newNode = new Node();
            newNode.data = i;
            newNode.edges = new Edge[0];
            newNode.edgeCount = 0;
            if (askName) {
                System.out.print("Enter name for person " + (i + 1) + ": ");
                newNode.name = scanner.next();
            } else {
                newNode.name = "Person " + (i + 1);
            }

            graph.nodes[i] = newNode;
        }
    }

    static void addEdge(Graph graph) {
        for (int i = 0; i < graph.numNodes; ++i) {
            for (int j = 0; j < graph.numNodes; ++j) {
                if (i == j) continue;

                System.out.print("Enter the amount " + graph.nodes[i].name
                        + " owes to " + graph.nodes[j].name + " (0 if none): ");
                int weight = scanner.nextInt();

                if (weight != 0) {
                    Edge newEdge = new Edge();
                    newEdge.from = i;
                    newEdge.to = j;
                    newEdge.weight = weight;

                    Edge[] newEdges = new Edge[graph.nodes[i].edgeCount + 1];
                    System.arraycopy(graph.nodes[i].edges, 0, newEdges, 0,
graph.nodes[i].edgeCount);
                    graph.nodes[i].edges = newEdges;

                    graph.nodes[i].edges[graph.nodes[i].edgeCount++] = newEdge;
                }
            }
        }
    }

    static void minCashflow(Graph graph, int n) {
        int[] netAmount = new int[n];
        int initial = 0;
        int minimized = 0;

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < graph.nodes[i].edgeCount; ++j) {
```

```java
                netAmount[i] += graph.nodes[i].edges[j].weight;
                netAmount[graph.nodes[i].edges[j].to] -= graph.nodes[i].edges[j].weight;
                initial++;
            }
        }
        System.out.println("\n\nInitial transactions:" + initial + "\n\n");

        while (true) {
            int maxCreditor = 0, maxDebtor = 0;
            for (int i = 1; i < n; ++i) {
                if (netAmount[i] > netAmount[maxCreditor]) maxCreditor = i;
                if (netAmount[i] < netAmount[maxDebtor]) maxDebtor = i;
            }

            if (netAmount[maxCreditor] == 0 && netAmount[maxDebtor] == 0)
                break;

            int minAmount = Math.min(-netAmount[maxDebtor], netAmount[maxCreditor]);
            netAmount[maxCreditor] -= minAmount;
            netAmount[maxDebtor] += minAmount;

            System.out.println(graph.nodes[maxDebtor].name + " pays " + minAmount
                    + " to " + graph.nodes[maxCreditor].name);
            minimized++;
        }
        System.out.println("\n\nMinimized transactions:" + minimized + "\n\n");
    }

    public static void main(String[] args) {
        Graph graph = new Graph();

        addNode(graph);

        addEdge(graph);

        minCashflow(graph, graph.numNodes);
    }
}
```

CODE IN C++

```cpp
#include <iostream>
#include<string>
using namespace std;

struct Edge {
    int from;
    int to;
    int weight;
};

struct Node {
    string name;
    int data;
    Edge* edges;
    int edgeCount;
};

struct Graph {
    int numNodes;
    Node* nodes;
};


void addNode(Graph* graph) {
    bool askName;
    cout << "Do you want to enter names for the persons? (1 for Yes, 0 for No): ";
    cin >> askName;

    cout << "Enter the number of persons: ";
    cin >> graph->numNodes;
    graph->nodes = new Node[graph->numNodes];

    for (int i = 0; i < graph->numNodes; ++i) {
        Node newNode;
        newNode.data = i;
        newNode.edges = nullptr;
        newNode.edgeCount = 0;

        if (askName) {
            cout << "Enter name for person " << i + 1 << ": ";
            cin >> newNode.name;
        }
        else {
            newNode.name = "Person " + to_string(i + 1);
        }

        graph->nodes[i] = newNode;
    }
}

void addEdge(Graph* graph) {
    for (int i = 0; i < graph->numNodes; ++i) {
        for (int j = 0; j < graph->numNodes; ++j) {
            if (i == j) continue;

            cout << "Enter the amount " << graph->nodes[i].name
                << " owes to  " << graph->nodes[j].name << " (0 if none): ";
            int weight;
            cin >> weight;

            if (weight != 0) {
                Edge newEdge;
                newEdge.from = i;
                newEdge.to = j;
                newEdge.weight = weight;

                Edge* newEdges = new Edge[graph->nodes[i].edgeCount + 1];
                for (int k = 0; k < graph->nodes[i].edgeCount; ++k) {
                    newEdges[k] = graph->nodes[i].edges[k];
                }
                delete[] graph->nodes[i].edges;

                newEdges[graph->nodes[i].edgeCount++] = newEdge;
                graph->nodes[i].edges = newEdges;
            }
        }
    }
}

void min_cashflow(Graph* graph, int n) {
    int* net_amount = new int[n]();
    int initial = 0;
    int minimized = 0;

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < graph->nodes[i].edgeCount; ++j) {
            net_amount[i] += graph->nodes[i].edges[j].weight;
            net_amount[graph->nodes[i].edges[j].to] -= graph->nodes[i].edges[j].weight;
            initial++;
        }
    }
    cout << "\n\nInitial transactions:" << initial << "\n\n";

    while (true) {
        int maxCreditor = 0, maxDebtor = 0;
        for (int i = 1; i < n; ++i) {
            if (net_amount[i] > net_amount[maxCreditor]) maxCreditor = i;
            if (net_amount[i] < net_amount[maxDebtor]) maxDebtor = i;
        }
        if (net_amount[maxCreditor] == 0 && net_amount[maxDebtor] == 0)
            break;

        int minAmount = min(-net_amount[maxDebtor], net_amount[maxCreditor]);
        net_amount[maxCreditor] -= minAmount;
        net_amount[maxDebtor] += minAmount;

        cout  << graph->nodes[maxDebtor].name << " pays " << minAmount
            << " to " << graph->nodes[maxCreditor].name << endl;
        minimized++;
    }
    cout << "\n\nMinimized transactions:" << minimized << "\n\n";

    delete[] net_amount;
}

int main() {
    Graph graph;


    addNode(&graph);

    addEdge(&graph);

    min_cashflow(&graph, graph.numNodes);


    delete[] graph.nodes;


    return 0;
}
```

Code in C#

```
using System;
using System.Collections.Generic;

public class Edge
{
	public int From;
	public int To;
	public int Weight;
}

public class Node
{
	public string Name;
	public int Data;
	public List<Edge> Edges = new List<Edge>();
}

public class Graph
{
	public List<Node> Nodes = new List<Node>();
}

public class Program
{
	public static void Main()
	{
		Graph graph = new Graph();

		AddNodes(graph);
		AddEdges(graph);
		MinimizeCashFlow(graph);
	}

	private static void AddNodes(Graph graph)
	{
```

```
            Console.Write("Do you want to enter names for the persons? (1 for Yes, 0 for No): ");
            bool askName = Console.ReadLine().Trim() == "1";

            Console.Write("Enter the number of persons: ");
            int numNodes = int.Parse(Console.ReadLine());

            for (int i = 0; i < numNodes; ++i)
            {
                Node newNode = new Node
                {
                    Data = i
                };

                if (askName)
                {
                    Console.Write($"Enter name for person {i + 1}: ");
                    newNode.Name = Console.ReadLine();
                }
                else
                {
                    newNode.Name = "Person " + (i + 1);
                }

                graph.Nodes.Add(newNode);
            }
        }

        private static void AddEdges(Graph graph)
        {
            for (int i = 0; i < graph.Nodes.Count; ++i)
            {
                for (int j = 0; j < graph.Nodes.Count; ++j)
                {
                    if (i == j) continue;

                    Console.Write($"Enter the amount {graph.Nodes[i].Name} owes to
{graph.Nodes[j].Name} (0 if none): ");
                    int weight = int.Parse(Console.ReadLine());

                    if (weight != 0)
                    {
                        Edge newEdge = new Edge
                        {
                            From = i,
                            To = j,
                            Weight = weight
                        };

                        graph.Nodes[i].Edges.Add(newEdge);
                    }
                }
            }
        }

        private static void MinimizeCashFlow(Graph graph)
        {
            int[] netAmount = new int[graph.Nodes.Count];
            int initial = 0;
            int minimized = 0;

            for (int i = 0; i < graph.Nodes.Count; ++i)
            {
                foreach (var edge in graph.Nodes[i].Edges)
                {
                    netAmount[i] += edge.Weight;
                    netAmount[edge.To] -= edge.Weight;
                    initial++;
                }
            }

            Console.WriteLine($"\n\nInitial transactions: {initial}\n");

            while (true)
            {
                int maxCreditor = 0, maxDebtor = 0;
                for (int i = 1; i < netAmount.Length; ++i)
                {
                    if (netAmount[i] > netAmount[maxCreditor]) maxCreditor = i;
                    if (netAmount[i] < netAmount[maxDebtor]) maxDebtor = i;
                }

                if (netAmount[maxCreditor] == 0 && netAmount[maxDebtor] == 0)
                    break;

                int minAmount = Math.Min(-netAmount[maxDebtor], netAmount[maxCreditor]);
                netAmount[maxCreditor] -= minAmount;
                netAmount[maxDebtor] += minAmount;

                Console.WriteLine($"{graph.Nodes[maxDebtor].Name} pays {minAmount} to 
{graph.Nodes[maxCreditor].Name}");
                minimized++;
            }

            Console.WriteLine($"\n\nMinimized transactions: {minimized}\n");
        }
    }
```

Code in C

```c
#include <stdio.h>
#include <stdlib.h>
struct Edge {
    int from;
    int to;
    int weight;
};
struct Node {
    char name[50]; // Assuming a maximum name length of 50 characters
    int data;
    struct Edge* edges;
    int edgeCount;
};
struct Graph {
    int numNodes;
    struct Node* nodes;
};
void addNode(struct Graph* graph) {
    int askName;
    printf("Do you want to enter names for the persons? (1 for Yes, 0 for No): ");
    scanf("%d", &askName);

    printf("Enter the number of persons: ");
    scanf("%d", &graph->numNodes);
    graph->nodes = (struct Node*)malloc(graph->numNodes * sizeof(struct Node));
    for (int i = 0; i < graph->numNodes; ++i) {
        struct Node newNode;
        newNode.data = i;
        newNode.edges = NULL;
        newNode.edgeCount = 0;
        if (askName) {
            printf("Enter name for person %d: ", i + 1);
            scanf("%s", newNode.name);
        } else {
            sprintf(newNode.name, "Person %d", i + 1);
        }
        graph->nodes[i] = newNode;
    }
}
void addEdge(struct Graph* graph) {
    for (int i = 0; i < graph->numNodes; ++i) {
      for (int j = 0; j < graph->numNodes; ++j) {
            if (i == j) continue;
            printf("Enter the amount %s owes to %s (0 if none): ", graph->nodes[i].name, graph-
>nodes[j].name);
            int weight;
            scanf("%d", &weight);
            if (weight != 0) {
                struct Edge newEdge;
                newEdge.from = i;
                newEdge.to = j;
                newEdge.weight = weight;
                graph->nodes[i].edges = realloc(graph->nodes[i].edges, (graph->nodes[i].edgeCount
+ 1) * sizeof(struct Edge));
                graph->nodes[i].edges[graph->nodes[i].edgeCount++] = newEdge;
            }
        }
    }
}

void minCashflow(struct Graph* graph, int n) {
    int* netAmount = (int*)malloc(n * sizeof(int));
    int initial = 0;
    int minimized = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < graph->nodes[i].edgeCount; ++j) {
            netAmount[i] += graph->nodes[i].edges[j].weight;
            netAmount[graph->nodes[i].edges[j].to] -= graph->nodes[i].edges[j].weight;
            initial++;
        }
    }
    printf("\n\nInitial transactions: %d\n\n", initial);
    while (1) {
        int maxCreditor = 0, maxDebtor = 0;
        for (int i = 1; i < n; ++i) {
            if (netAmount[i] > netAmount[maxCreditor]) maxCreditor = i;
            if (netAmount[i] < netAmount[maxDebtor]) maxDebtor = i;
        }
        if (netAmount[maxCreditor] == 0 && netAmount[maxDebtor] == 0)
            break;
        int minAmount = netAmount[maxDebtor] < -netAmount[maxCreditor] ? netAmount[maxDebtor] : -
netAmount[maxCreditor];
        netAmount[maxCreditor] += minAmount;
        netAmount[maxDebtor] -= minAmount;

        printf("%s pays %d to %s\n", graph->nodes[maxDebtor].name, minAmount, graph-
>nodes[maxCreditor].name);
        minimized++;
    }
    printf("\n\nMinimized transactions: %d\n\n", minimized);
    free(netAmount);
}
int main() {
    struct Graph graph;

    addNode(&graph);
    addEdge(&graph);
    minCashflow(&graph, graph.numNodes);
    free(graph.nodes);


    return 0;
}
```

Code in Python

```python
N = 100
graph = [[0] * N for _ in range(N)]
people = []
n = 0

def get_min(a):
    return min(range(n), key=lambda i: a[i])

def get_max(a):
    return max(range(n), key=lambda i: a[i])

def min_cashflow_rec(amt):
    mcr = get_max(amt)
    mdb = get_min(amt)
    if amt[mcr] == 0 and amt[mdb] == 0:
        return
    minval = min(-amt[mdb], amt[mcr])
    amt[mcr] -= minval
    amt[mdb] += minval
    print("\n{} will pay {} to {}".format(people[mdb], minval, people[mcr]))
    min_cashflow_rec(amt)

def min_cashflow(graph):
    global n
    amt = [0] * n
    for p in range(n):
        amt[p] = sum(graph[i][p] - graph[p][i] for i in range(n))
    min_cashflow_rec(amt)

if __name__ == "__main__":
    print("***** Welcome to CashU - settle debts with less. *****")
    n = int(input("\nEnter no. of people: "))

    people = [input(f"Enter name#{i + 1}: ") for i in range(n)]

    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            graph[i][j] = int(input(f"How much {people[i]} has to pay {people[j]}?   "))

    min_cashflow(graph)
    print("\nPress 0 to exit...")
    t = int(input())
    if t == 0:
        exit(1)
```

Conclusion

Cash Flow Minimizer project offers an efficient solution for optimizing financial transactions within a group. Utilizing a greedy algorithm and a graph data structure, the program minimizes the number of transactions required to settle debts. Implemented in various languages, including Java, C++, C#, C, R, and Python, the project demonstrates versatility. While successful in achieving its goals, further refinement and feature additions could enhance its real-world applicability. Overall, the Cash Flow Minimizer project effectively combines algorithmic principles and data structures to address practical challenges in financial management.

Sources

- Geeksforgeeks
- Chat gpt
- Black box
- Multiple sources from github