SPLITZI - CASH FLOW APPLICATION

A PROJECT REPORT

SUBMITTED IN COMPLETE FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF
BACHELOR OF TECHNOLOGY
IN
ENGINEERING PHYSICS

Submitted by:

HEMANG SINHA
(2K19/EP/039)

PRIYANSH TYAGI
(2K19/EP/075)

UNDER THE SUPERVISION OF
Asst. Prof. Anukriti Kaushal

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(FORMERLY Delhi College of Engineering)
Bawana Road, Delhi-110042

NOVEMBER 2021

CANDIDATE'S DECLARATION

We, Hemang Sinha (Roll No. 2K19/EP/039) and Priyansh Tyagi (Roll No. 2K19/EP/075), students of B. Tech. Engineering Physics, hereby declare that the project dissertation titled "SPLITZI - Cash Flow Application" which is submitted by me to the Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of Bachelor of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma, Fellowship or other similar title or recognition.

Place: DTU, Delhi

Hemang Sinha (2K19/EP/039)

Date: 22nd November' 2021

Priyansh Tyagi (2K19/EP/075)

CERTIFICATE

I hereby certify that the project dissertation titled “SPLITZI - Cash Flow Application” which is submitted by Hemang Sinha (2K19/EP/039) and Priyansh Tyagi (2K19/EP/075) to the Department of Computer Science & Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of Bachelor of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date: 22nd November’ 2021

Anukriti Kaushal

SUPERVISOR

ACKNOWLEDGEMENT

We are very thankful to Dr. Rajni Jindal (Head of Department, Computer Science & Engineering Department) and Ms. Anukriti Kaushal (Assistant Professor, Computer Science & Engineering Department) and all the faculty members of the Computer Science & Engineering Department of DTU for providing immense support and guidance for the completion of the project undertaken by us. It is with their supervision that this work came into existence.

We would also like to express our gratitude to the university for providing the laboratories, infrastructure, test facilities and environment which allowed us to work without any obstructions.

We would also like to appreciate the support provided by our lab assistants, seniors and peer group who aided us with all the knowledge they had regarding various topics.

CONTENTS

LIST OF FIGURES

INTRODUCTION

Project Background

Cash flow can be defined as the net amount of cash and cash equivalents carried into and out of a company. The number of transactions between persons is growing at an exponential rate these days. As the number of transactions increases, so do the transaction fees we pay to our banks and brokers. The costs of buying or selling a product or service are known as transaction costs. And, in order to keep these transactions running correctly, a lot of money is spent on power and employees.

A technology that helps us decrease the number of transactions between various persons while also saving resources is urgently needed. It's a challenging effort to manage and handle these transactions while also offering an elegant solution that is both correct and reduces the number of transactions. As a result, we've created a method to assist us in reducing these cash flow transactions and conserving our essential resources.

It's critical to treat these cash flow transactions with care and precision. Any inconsistency between these transactions might imperil a large number of futures contracts. As a result, we've made every effort to reduce the possibility of errors.

Problem Statement

We started with a simple math problem but in order to devise this algorithm we have explored NP-completeness problems which deal with computational complexity theory.

For any given transactions, we have to always check for the scenario of each Giver making only one transaction in order to settle his/her debts. So, in the optimal scenario, each Giver will transfer money to exactly one Receiver. Thus, every Receiver should either receive the entire money from a Giver or not accept any amount from them altogether.

Thus finally, in technical terms we can say that, given a Directed Graph representing transactions, change (if needed) the weights on the existing edges without introducing newer ones.

Objective of the Project

Our aim is to develop a platform which uses an algorithm to restructure cash transactions within groups of people. The algorithm does not aim to change the total amount that any

of the participants owes, but it makes it easier to pay people back by minimizing the total number of transactions.

Thus we started with the below listed objectives,

- Developing an optimized algorithm for minimized cash flow transactions.
- Building a user-friendly web interface which can be used by people to visualize the efficient cash flow method.

THEORY

Important Terminologies

1. Graphs

Graphs are two-dimensional structures made up of vertices and edges that connect them. A graph which can be written as G = (V, E) consists of,

- V, a nonempty set of vertices (also known as nodes), and
- E, a set of edges

Endpoints are either one or two vertices that are associated with each edge. An edge is defined as a connection between two points. The different types of graph are as follows,

a. Directed Graph: A directed graph is made up of a set of vertices (nodes) linked by edges, each with its own direction. The edges of a graph are commonly depicted by arrows pointing in the direction of traversal.

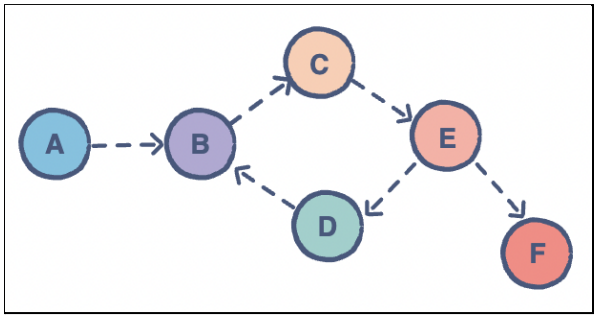

Figure.1. Illustration for Directed Graph

b. Undirected Graph: The edges of an undirected graph are bidirectional and have no associated direction. As a result, the graph may be traversed both ways. The fact that there is no arrow indicates that the graph is undirected.

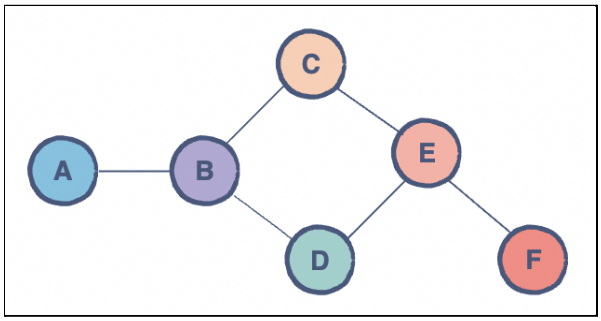

Figure.2. Illustration for Undirected Graph

2. **Breadth-First Search (BFS)**

   For graphs, when a dead-end occurs in any iteration, the Breadth-First Search (BFS) method traverses a network breadthward and utilizes a queue to remember to retrieve the next vertex to start a search.

3. **Depth-First Search (DFS)**

   For graphs, the Depth-First Search (DFS) is a recursive technique that is used to traverse or explore data structures such as trees and graphs. The algorithm starts from the root node (in the case of a graph, any random node can be used as the root node) and examines each branch as feasible before backtracking.

4. **Heap**

   A Heap is a Tree-based data structure with a full binary tree as the tree. Heaps can be divided into two categories,

   Min-Heap: In a Min-Heap, the root node's key must be the smallest of all the keys present in all of its descendants. For all subtrees of that Binary Tree, the same property must be true recursively.

   Max-Heap: In a Max-Heap, the root node's key must be the largest of all the keys present in all of its descendants. For all subtrees of that Binary Tree, the same property must be true recursively.

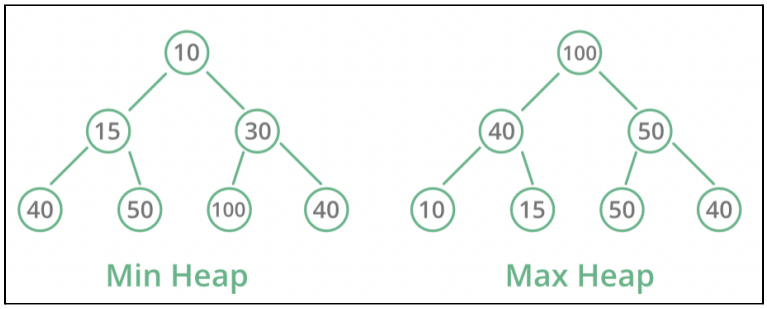

Figure.3. Illustration for Min Heap and Max Heap

5. **Heapify**

   Heapify is the act of rearranging a heap in order to keep the heap property: the root node's key is more extreme (greater or less) than or equal to the keys of its children. If the root node's key isn't more extreme, swap it with the child with the most extreme key, then heapify that child's subtree recursively. To begin, the child subtrees must be heaps.

6. **Hash-Table or Hash-Map**

   A hash table or hash map is a data structure that uses an associative array abstract data type to map keys to values. A hash table employs a hash function to generate an index, usually referred to as a hash code, into an array of buckets or slots from which the necessary data may be retrieved. The key is hashed during lookup, and the resultant hash shows where the relevant value is kept.

7. **NP-Completeness**

   In computational complexity theory, a problem is NP-complete (nondeterministic polynomial-time complete) when it is a problem for which the correctness of each solution can be verified quickly and a brute-force search algorithm can actually find a solution by trying all possible solutions and, the problem can be used to simulate every other problem for which we can verify quickly that a solution is correct. In this sense it is the hardest of the problems to which solutions can be verified quickly so that if we could actually find solutions of some NP-Complete problem quickly, we could quickly find the solutions of every other problem to which a solution once given is easy to check.

NP - Completeness Proof

First we must reformulate the given problem as a decision problem.

Let G = (V, E) be a weighted directed graph. Given G and k, return whether there exists an equivalent graph G' with at most k edges.

Given a G′, we can verify it's valid and has at most k edges in polynomial time. So the Splitzi algorithm is in NP.

Next, construct a reduction from Partition to Splitzi algorithm. Given a list S of positive integers, we want to know if there is a partition S_1,S_2 such that both sets have the same total sum.

Given $S=\{a_1,a_2,\ldots,a_n\}$, construct G with n+2 vertices $v_1,v_2,\ldots,v_{n+2}$. Add edges such that v_i owes a_i, and v_{n+1},v_{n+2} request $\frac{1}{2}\sum_{i=1}^{n} a_i$ each.

Any solution to this instance must have at least n transactions, because n people owe money. Furthermore, if the solution has exactly n transactions, v_i must send a_i for their transaction. The edges entering v_{n+1} and v_{n+2} correspond exactly to a partition of

$\{a_1,\ldots,a_n\}$ Thus, there is a solution with n transactions if and only if a partition exists. Since the partition is NP-Complete, it follows that our Splitzi algorithm is NP-Complete.

Algorithm

The problem statement in technical terms we were dealing with is stated as, given a Directed Graph representing transaction, change (if needed) the weights on the existing edges without introducing newer ones. The algorithm that we have developed aligned to the given problem statement is described below,

1. Feed the debts in the form of a directed graph (let's represent it by G) to the algorithm.
2. Select one of the non-visited edges, say (u, v) from the directed graph G.
3. Now with u as source and v as sink, run a maxflow algorithm (possibly Dinic's maxflow algorithm, since it's one of the optimal implementations) to determine the maximum flow of money possible from u to v.
4. Also, compute the Residual Graph (let's represent it by G'), which indicates the additional possible flow of debts in the input graph after removing the flow of debts between u and v.
5. If maximum flow between u and v (let's represent it by max-flow) is greater than zero, then add an edge (u, v) with weight as max-flow to the Residual Graph.
6. Now go back to Step 1 and feed it the Residual Graph G'.
7. Once all the edges are visited, the Residual Graph G' obtained in the final iteration will be the one that has the minimum number of edges(i.e. transactions).

It outputs the final sequence of transactions to be made after simplifying debts. The algorithm has a complexity of $O(V^2E^2)$, where V is the number of vertices in the graph and E is the number of edges in the graph. Here, $O(V^2E)$ is the complexity of the Dinic's implementation of the Maximum flow Problem and the extra O(E) is due to the algorithm iterating over all the edges in the graph.

PROJECT DEVELOPMENT

Flowchart of Program Execution

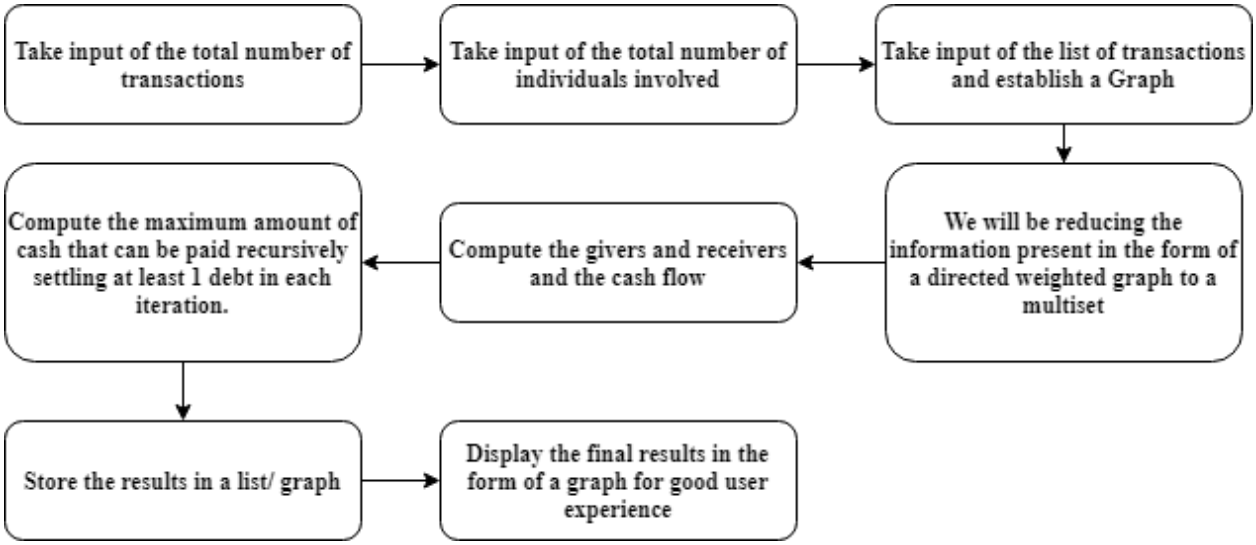

Figure.4.: Flowchart of Program Execution

Technologies Used

1. **C++:** It is undoubtedly one of the most powerful and oldest programming languages that still reigns supreme in the programming world. For our project, we utilized C++ as the programming language to design and test the method and to develop the prototype of the algorithm for our project.

2. **JavaScript:** JavaScript, sometimes known as JS, is a computer language that follows the ECMAScript standard. JavaScript is a high-level programming language that is frequently compiled just in time and is multi-paradigm. With the help of features like first-class functions, dynamic typing, and prototype-based object orientation, to name a few, we aimed to design a primary, minimalistic user interface that can be implemented for web deployment.

3. **HTML:** HTML, or HyperText Markup Language, is the industry standard markup language for web-based writing and thus proved to be the basic building block for our web-based project.

4. **CSS:** CSS (Cascading Style Sheets) is a markup language for specifying the appearance of a document written in HTML. CSS has been well utilized by us to style the pages of our web project.

PROJECT SCREENSHOTS AND EXECUTION DETAILS

We used web technologies like HTML, CSS and JavaScript to develop the interface shown in Figure.4. below,

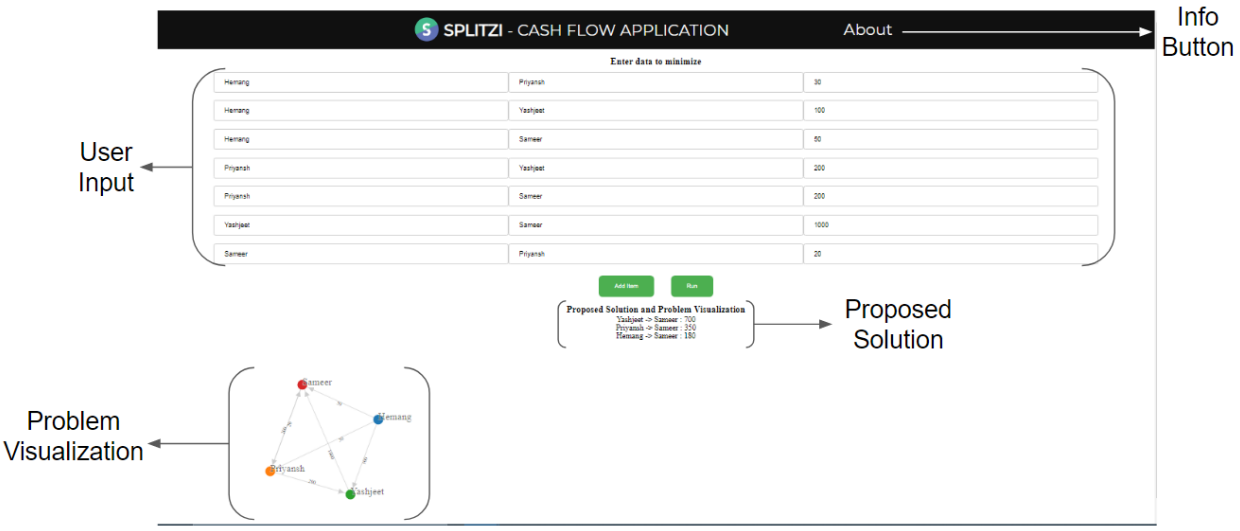

Figure.5. Landing Page

The landing page has the below listed sections/ components,

1. **User Input Section:** The User Input Section consists of 3 input fields namely, *'FROM'*, *'TO'* and *'AMOUNT'*, for money lender, money receiver and transaction amount respectively. The first two fields *'FROM'* and *'TO'* accept a String input whereas the third field *'AMOUNT'* accepts a Floating-point literal. Data entered in these fields is used to develop a Graph for the entered problem statement.

2. **Add Item Button:** Pressing the Add Item Button adds a new row in the interface and prompts the user to make a new entry for the problem statement.

3. **Problem Visualization Section:** On pressing the Run Button the JavaScript code responsible for building the graph is called which utilizes the information supplied to the interface via the User Input Section fields, processing the data taking care of any probable exception and returns an SVG image of a weighted directed graph, with names of the individuals involved in the transaction as nodes and the transaction amount as the weight of individual edge.

4. **Proposed Solution Section:** The Proposed Solution Section is situated below the Add Item Button and Run Button. The result of the optimized solution generated for the submitted problem statement is shown in this section formulated from the algorithm about which we have discussed in the previous sections.

5. **Run Button:** Clicking the Run Button triggers the JavaScript algorithm developed in the backend to initiate the process to develop the problem visualization SVG image which is situated in the Proposed Solution Section.

6. **About/ Info Button:** The About Button located in the top Navigation Bar when clicked opens a new page in the web browser with a brief description of the project and details of the developers of the project as shown in Figure.5. below.

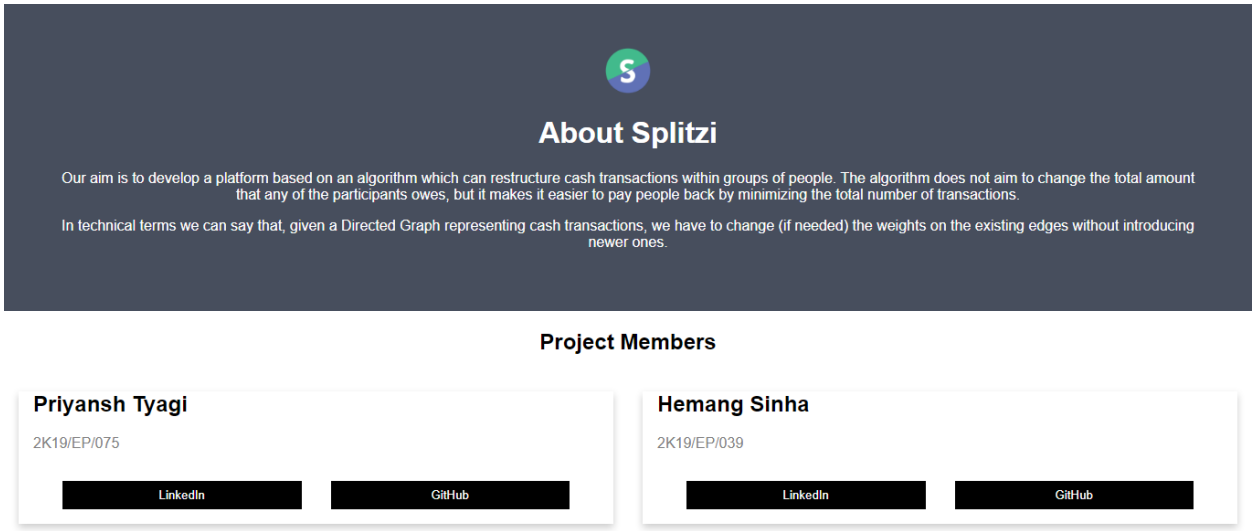

Figure.6. About Splitzi Page

The About Splitzi Screen shown in Figure.5. consists of a brief description of the project along with contact details of the project members.

Click here to see a video demonstration of this project.

IMPORTANT LINKS

1. Video demonstration - https://drive.google.com/file/d/1Vi1gWL30JEiOCNCS5EqHR84TTflPrHFs/view?usp=sharing
2. Code repository (GitHub Link) - https://github.com/priyanshty19/DS-MTE-Splitwise
3. Commit history (GitHub Commit) - https://github.com/priyanshty19/DS-MTE-Splitwise/commits?author=priyanshty19

REFERENCES

1. An Introduction to Data Structures and Application by Jean Paul Tremblay & Pal G. Sorenson (McGraw-Hill Education)
2. Data Structures Using C by Reema Thareja (Oxford University Press)
3. Object Oriented Programming in C++ by E. Balaguruswamy (McGraw-Hill Education)
4. https://medium.com/@mithunmk93/
5. OPENGENUS - https://iq.opengenus.org/algorithm-behind-bill-splitting-app/
6. Work @ Tech - https://workat.tech/machine-coding/editorial/how-to-design-splitwise-machine-coding-ayvnfo1tfst6
7. Coders Packet - https://coderspacket.com/splitwise-in-c
8. https://en.wikipedia.org/wiki/NP-completeness