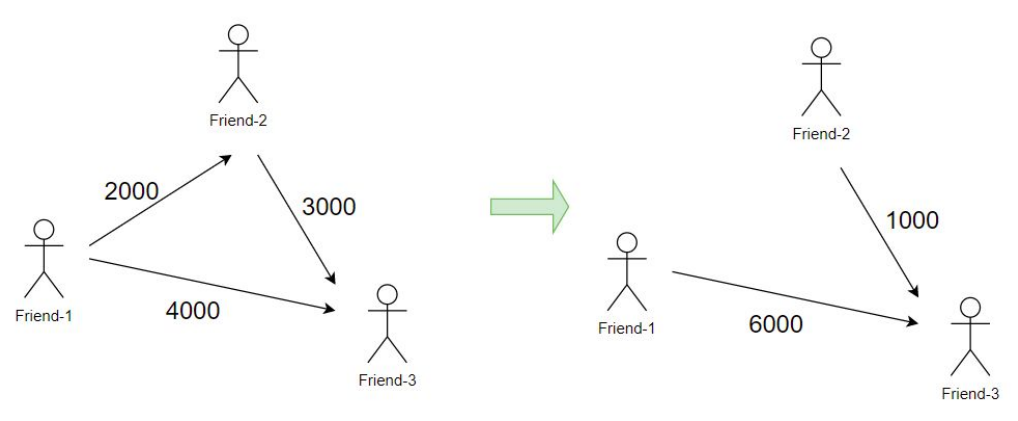
Innovation Design - Group Debt Simplification

Ammar Ahmad-2201AI04
Anand Kumar-2201AI05

Under Dr. Arijit Mondal

Introduction

This problem involves fairly dividing shared expenses among a group of friends and minimizing the number of transactions required to settle all debts. For example, if three friends go on a trip and pay for different expenses, each person should contribute equally. The problem is to calculate how much each person owes or is owed, and then determine the fewest transactions needed to balance the amounts. Instead of having everyone pay multiple people, the solution seeks to simplify it to fewer payments.



Problem Statement

Mathematical Representation of the problem:

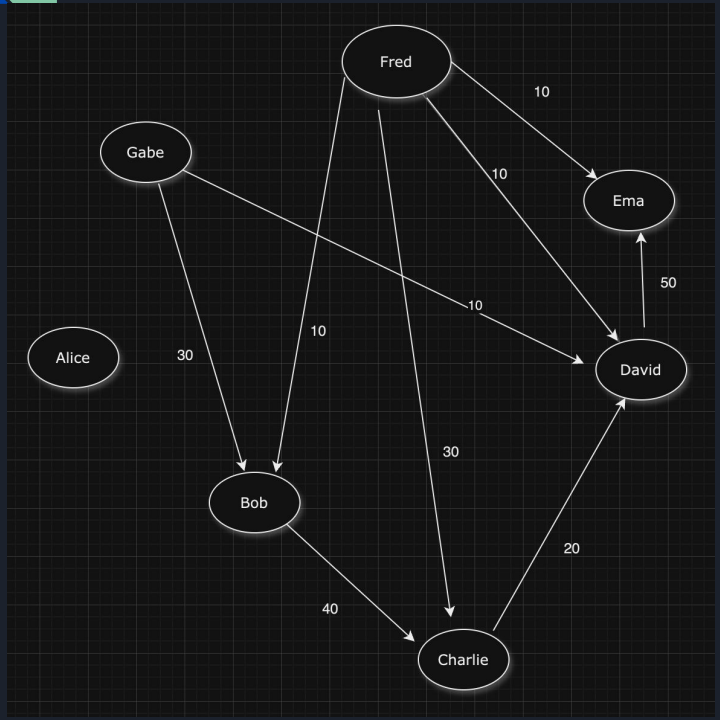
Consider a group of n participants P={P1,P2,...,Pn}, where each participant may either owe or be owed money by one or more other participants. Let d_{ij} denote the debt amount that participant P_i owes to participant P_j. The debts d_{ij} form a directed weighted graph G=(P,E), where E is the set of directed edges, and the weight of edge $e_{ij} \in E$ is the amount d_{ij}.

The goal is to restructure the debt relationships to minimise:

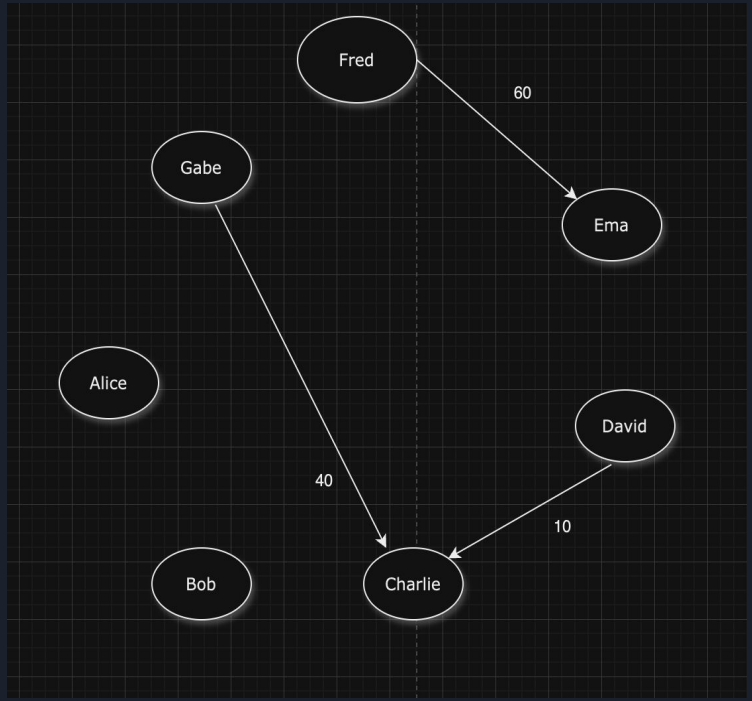
- The total number of transactions(the number of edges-- #d where d_{ij} != 0) in the new graph (AND/OR)
- The total transaction volume(sum of all the weights of edges - $\sum d_{ij}$) in the new graph

required to settle all debts while ensuring that each participant's net balance remains unchanged. Specifically, we seek to find a new graph G' = (P, E') such that net balance for each person is preserved.

How Initial and final transactions will look like?



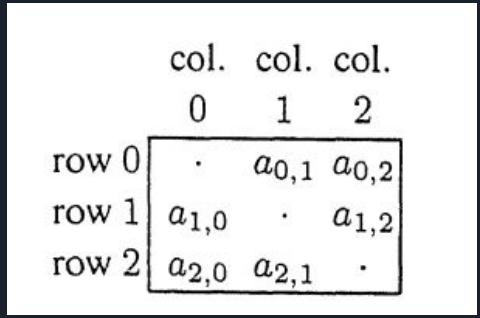
Initial Transactions(8)



Final Transactions(3)

Analysis

- For N <= 2, the problem is trivial and one to one settling is enough.
- For N > 2, We can start with a N x N debt matrix such as:



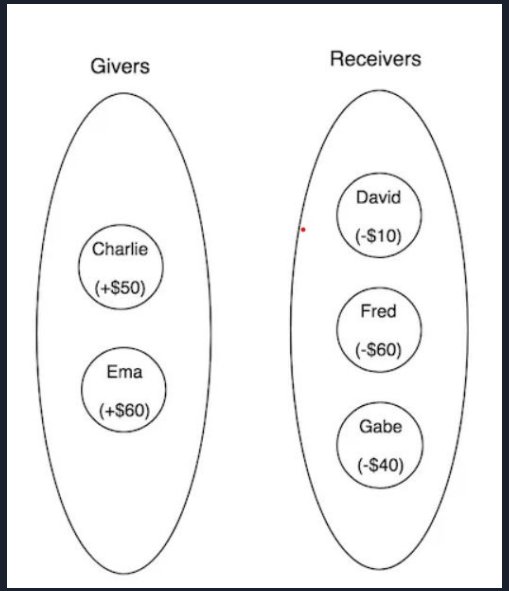
	col. 0	col. 1	col. 2
row 0	·	$a_{0,1}$	$a_{0,2}$
row 1	$a_{1,0}$	·	$a_{1,2}$
row 2	$a_{2,0}$	$a_{2,1}$	·

- Equivalently the solution will be a new N x N matrix that minimises the number of non zero values while also maintaining initial sum of all columns and all rows.
- Therefore a trivial solution will have $(N^2-N)/2$ number of transactions.
- A Practical solution can reduce this upto N transactions. This can be done by introducing a central account which collects all the debts at the end of each time period(say year) and distributes it among the creditors by maintaining a track of total debt and credit for each person.
- A more optimal way can guarantee no more than N-1 transfers.

The Greedy Solution

- **The greedy solution can guarantee a solution of at max N-1 transfers.**
- **Here's how greedy works:**

We calculate the net credit/debit for each person and sort them in order.



The Greedy Solution

- **We sort the net credit values in increasing order.**

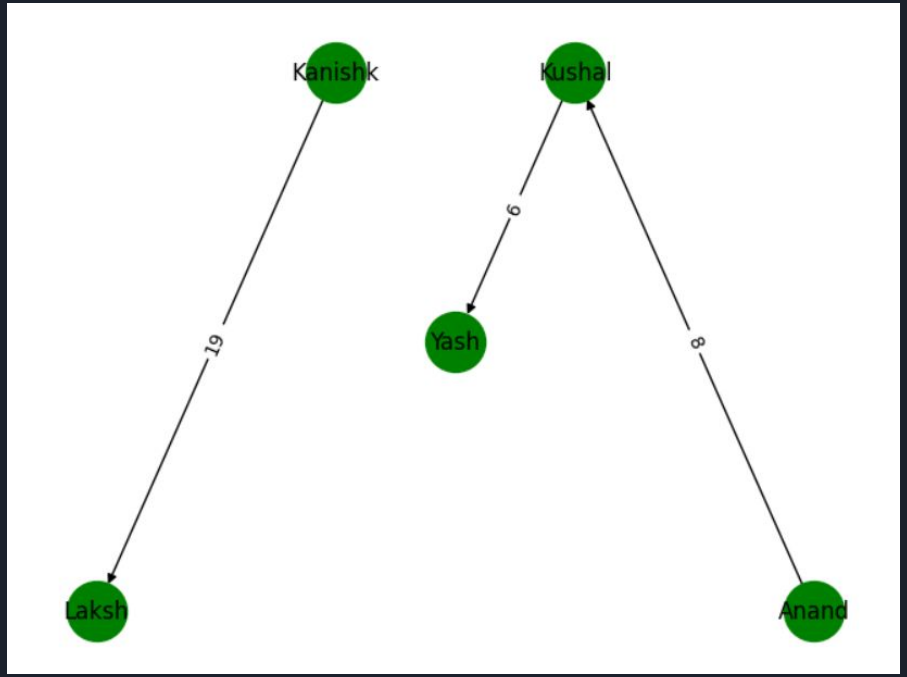
```
people = ["Laksh", "Kushal", "Anand", "Yash", "Kanishk"]
debts = [
    ("Laksh", "Kushal", 5),
    ("Laksh", "Anand", 3),
    ("Kushal", "Laksh", 2),
    ("Kushal", "Kanishk", 5),
    ("Anand", "Laksh", 10),
    ("Anand", "Yash", 4),
    ("Anand", "Kanishk", 6),
    ("Anand", "Kanishk", 2),
    ("Yash", "Kushal", 4),
    ("Kanishk", "Laksh", 15),
    ("Kanishk", "Yash", 6),
    ("Kanishk", "Anand", 11),
]
```

Credits are as follows:

- `{'Laksh': 19, 'Kushal': 2, 'Anand': -8, 'Yash': 6, 'Kanishk': -19}`
- `{'Kanishk': -19, 'Anand': -8,  'Kushal': 2, 'Yash': 6, 'Laksh': 19}`
- And then we can have transactions between the highest debtor with the highest creditor. The remaining amount is added back to the priority queue.
- Greedy Solution will therefore have at max N-1 transactions.

The Greedy Solution

Therefore this is the greedy solution which minimises the number of transactions to 3 (also the most optimal solution).



Another Way: N-1 transactions

We know that the total balance of the group is 0, so Grace's balance must be the opposite of the sum of everybody else's balance, or, denoting each person's balance as b_p,

$$b_{\text{Grace}} = -\sum_{p \in \text{group},\, p \neq \text{Grace}} b_p.$$

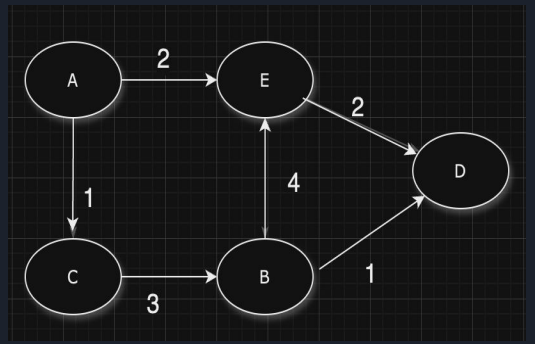
But $-\sum_{p \in \text{group},\, p \neq \text{Grace}} b_p$ is exactly the net amount of money Grace is receiving, because she has one transaction for each other person in the group, worth exactly their balance.

SInce the solutions satisfies our balance constraints, so it's an allowable way for the group to resolve their debt with utmost N-1 transactions.

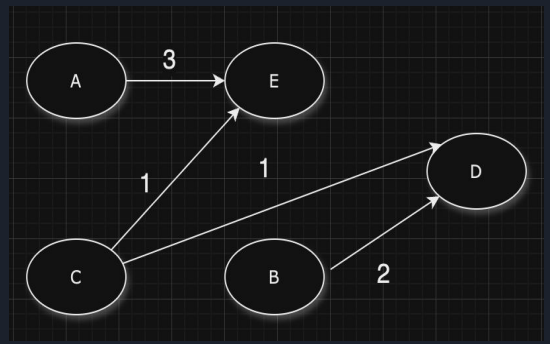
Greedy solution and How does it Fail?

The greedy approach focuses on always settling the maximum possible debt in each step, where the friend who owes the most (maximum debtor) pays the friend who is owed the most (maximum creditor). This process is repeated until all debts are settled.

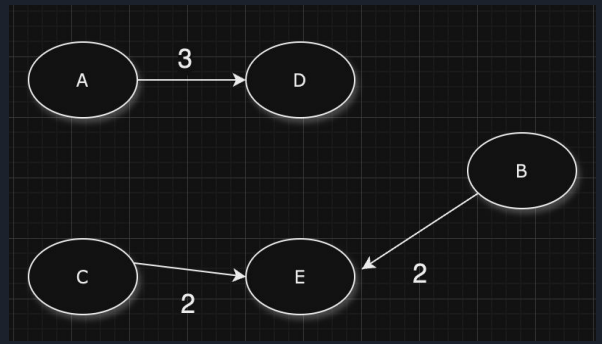
Example where greedy fails:-



Initial Transactions(6)



Greedy Solution(4)



Optimal Solution(3)

Therefore.....

Therefore we gave a thought about whether we can find the optimal solution to the problem in polynomial time at all. And therefore we moved on to proving that the problem is NP Complete.

We have concluded the following:

- The Problem is NP Complete and a polynomial time solution doesn't exist as of now.
- This problem can be proven NP Complete by reducing a known NP complete such as 3 Partition Problem or Subset Sum Problem to our problem.
- For convenience we chose maximum zero sum subset.

Methodology

- **Problem Representation**
  - The shared expense scenario is modeled as a directed weighted graph, where:
    - Nodes represent individuals.
    - Edges represent debts, with weights corresponding to the owed amounts.

- **Proving NP-Completeness**
  - Reducing a known NP-complete problem, the Maximum Zero-Sum Partition Problem (MZSP), to Debt Minimization Problem, demonstrating that Debt Minimization Problem is at least as hard as MZSP

- **Approximation Algorithm Design**
  - To address the complexity, we have tried implementing **approximation algorithms** to find the best possible heuristic. This algorithm aims to produce near-optimal solutions with a significant reduction in computational effort.

- Validation and Testing
  - The algorithms are tested on various synthetic and datasets, and their performance is compared based on number of transactions and its time complexity

Extending the N-1 Idea

- From the 3 partition problem, and extending the greedy solution of n-1 transactions. We can find the most optimal solution.
- We will start with finding the net credits/debits for each person.
- Say,

  A : -3 B : -2 C:-2 D: +3 E: +4

- Now we can prove that the minimal number of transactions in a group settlement can not be less than N-1 if no component can individually settle their transactions each.
- Therefore, Our task is to partition the N people into maximum number of subsets such that each subset satisfies the sum zero constraint.
- Let's say We have a group of people with credits:

  2 0 -1 1 -1 -1

- We will divide the group into as many subsets as possible with sum zero each.

Extending the N-1 Idea

- To Prove why this is optimal:
- As for each subset with no more components possible, minimum number of transactions will be n'-1 where n' is the number of people in that group.
- Say we divide our group into k groups with sum 0 each i.e.
- {{2, -1, -1} {0} {1, -1}}
- The total number of transactions will be (n1'-1) + (n2'-1) + (n3'-1) + ….
- We can write this as n1'+n2'+n3'+....-k
- We can replace n1'+n2'+n3'+.... With the initial number of people in the group.
- Therefore the transactions will be N-k.
- So if we maximise the number of partitions with sum zero each. We will find the most optimal solution.

Our Task therefore is: given a list of debits/credits: Partition them into as many subsets with sum zero each. This Problem can be proven NP complete.

Proving NP-Completeness

We start by

- **defining the problem statement.**
- **Showing that the problem is NP.**
- **We use a known NP Hard problem (Maximum Zero Sum Partition) and reduce it to our problem.**
- **Therefore we can conclude our problem is also NP Complete.**

Link: NP Complete Proof

Heuristic Based Solutions:

We have tried implementing various heuristics to find out which one is best approximation for this problem statement.

Observations

- **Greedy Algorithm**
  - Matching the Greatest debt with greatest credit.
  - Time complexity of this algorithm is nlogn where n is the number of people.

- **Linear Programming Algorithm**
  - Use a linear programming solver to approximate the minimum number of transactions.
  - Time Complexity is O(n^3) where n is the number of people.

- **Closest Match Algorithm**
  - Finding a closest match creditor for an equivalent debt amount Set constraints to ensure that all debts are settled.
  - The time complexity of the closest_match_heuristic function is O(n^2 log n).

- **Auction Based Approximation**
  - Finding a closest match creditor for an equivalent debt amount Set constraints to ensure that all debts are settled.
  - The time complexity of the closest_match_heuristic function is O(n^2 log n)

- **Randomized Rounding Approximation**
  - Use probabilistic techniques to explore potential solutions and converge to a near-optimal one.Use optimization methods (e.g., annealing) to iteratively improve the solution.
  - The time complexity of the randomized_rounding_approximation function is O(n^2)

:

- **Network Flow Approximation**
  - Model the debt settlement problem as a flow network, connecting debtors to creditors with edges representing possible transactions, and calculate the maximum flow to determine optimal transactions.
  - The time complexity of the network_flow_approximation function is O(m^3 * n^2), where m is the number of debtors and n is the number of creditors in the balances list.
- LP using pulp Approximation
  - Variation for the linear programming solution. Uses pulp. Tries minimising the number of transactions.
  - The time complexity of the minTX_lp function is O((m * n)^3)

Demonstration

We have tried implementing the brute force and various approximation algorithms on python. To do this we have used python libraries like matplotlib and networkx to visualise our transactions cases.
Google Colab Link: ColabLink

Aim 2

Minimising the amount of cash volume to settle all the debts.

We have also observed that a greedy solution in a smallest zero subset will not just make sure of n'-1 transactions but also minimise the cash volume among the n' people. Therefore minimising the cash volume is a trivial problem. And we can apply greedy at each minimal subset to get to the optimal solution.

- {'Kanishk': -19, 'Anand': -8, 'Kushal': 2, 'Yash': 6, 'Laksh': 19}
- In this problem, if we divide into subsets:
- {'Laksh': 19, 'Kanishk': -19}
- {Kushal: 2, 'Anand': -8, 'Kushal': 6}
- We can apply greedy in each subset to get a

Total transaction volume to be sum of all the debts

In any case, minimal cash volume will always be

Sum of debts or -(Sum of credits)

```
people = ["Laksh", "Kushal", "Anand", "Yash", "Kanishk"]
debts = [
    ("Laksh", "Kushal", 5),
    ("Laksh", "Anand", 3),
    ("Kushal", "Laksh", 2),
    ("Kushal", "Kanishk", 5),
    ("Anand", "Laksh", 10),
    ("Anand", "Yash", 4),
    ("Anand", "Kanishk", 6),
    ("Anand", "Kanishk", 2),
    ("Yash", "Kushal", 4),
    ("Kanishk", "Laksh", 15),
    ("Kanishk", "Yash", 6),
    ("Kanishk", "Anand", 11),
]
```

Conclusion

- **Greedy Algorithm** is optimal in terms of **time complexity**, with a time complexity of **O(n log n)**, but it is not optimal in terms of minimizing the **number of transactions**.

- **Linear Programming / Auction Based** provides a more **optimal solution** with respect to minimizing the **number of transactions**, but it has a **higher time complexity O(n^3)** and is less efficient than the greedy approach in terms of computational resources.

Future Plans

1. **Scaling Approximation Algorithms**
    - Enhance the approximation techniques to make the solution adaptable for general use cases.
2. **Developing a Scalable Mobile App/Website**
    - Design and implement a mobile application capable of solving group expense problems in polynomial time, offering a seamless and practical tool for managing shared expenses.

References and Links

- Verhoeff, Tom. “Settling multiple debts efficiently: An invitation to computing science.” Informatics in Education-An International Journal 3.1 (2004): 105-126

- Guillaume Fertin, Oscar Fontaine, Géraldine Jean, Stéphane Vialette. The Maximum Zero-Sum Partition Problem. 25th International Computer Symposium, ICS 2022, Dec 2022, Taoyuan, Taiwan. pp.73-85, ff10.1007/978-981-19-9582-8_7ff. Ffhal-04293802f

- “Introduction to Algorithms, Third Edition” by Thomas H. Cormen and Charles E. Leiserson.