



SPLITCOIN

Data Structures Project

Submitted To:

Ms Anuja Arora

Submitted by:

19103154 Kriti Swaroop B5

19103175 Khushi Jain B5

ABSTRACT

Keeping tabs of your credits and debts has always been a difficult job. Often people keep a notebook and perform difficult calculations, which is susceptible to error. In the advent of technology, it is better for us to let computers do such menial work for better results and saving time simultaneously.

We, as college students have experienced this ourselves in our daily lives. Being hostellers, we have to borrow money from our friends many times and it is difficult to keep note of every transaction.

Our aim in this project is to make this process error free and time saving. We simplify the problem by making a program which minimizes the number of transactions involved.

INTRODUCTION

Splitcoin minimizes the total number of transactions made within a group. It asks the user for the name of the creditor, debtor and the debt amount and outputs the minimized number of transactions.

TECHNOLOGIES USED

Data Structure: Graph

Algorithm: Dinic's maximum-flow algorithm

STL has been extensively used in the project

- Vector
- Map
- Set
- Queue
- Tuple
- File handling – To store the transactions

CONCEPTS

A ***flow network*** is a directed graph where each edge has a certain capacity which represents the maximum amount of flow the edge can receive.

1. The amount of flow running through an edge cannot exceed this capacity.

2. Incoming flow is equal to outgoing flow for every node except the source S and sink T.

Initially, the flow through each edge is 0 and capacity is a nonnegative value.

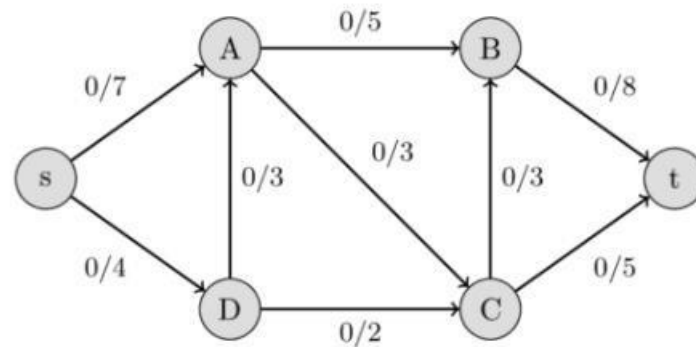


Figure 1 – Directed graph representing a flow network

The **maximum flow problem** calculates the maximum amount of flow out of the source node and into the sink node.

To find the maximum flow between the source and the sink node, following algorithms are used:

1. Dinic's Algorithm
2. Ford Fulkerson Algorithm
3. The Edmonds–Karp algorithm, etc

Since Dinic's algorithm is a strongest polynomial algorithm for maximum flow with a time complexity of $O(V^2E)$.

An **augmenting path** is a path of edges from source to sink in the residual graph in which each edge has a remaining capacity (Initial capacity - flow) greater than 0.

Augmenting the flow means updating the flow values of the edges along the augmenting path. For forward edges, this means increasing

the flow by the *bottleneck value* (edge of the path with smallest capacity).

While augmenting the flow along the augmenting path we need to decrease the flow along each residual edge by the bottleneck value in the residual graph. This is done to undo bad augmenting paths which do not lead to maximum flow.

Dinic's algorithm is used repeatedly to find the maximum flow between each pair of vertices.

DESCRIPTION

Header Files used:

```
#include<iostream>
```

```
#include<vector> #include<map>
```

```
#include<queue>
```

```
#include<set>
```

CLASS Edge

This class represents an edge that connects two nodes with a given capacity. '*from*' stores the debtor, '*to*' stores the creditor, '*flow*' gives the flow running through the edge at the moment. '*capacity*' is the maximum flow possible between the two nodes and lastly '*rev*' points to the edge in the reverse direction.

CLASS Graph

This builds a graph, hooking the nodes with edges and contains a number of functions used in the project.

METHODS USED

void addEdge (int, int, int) - This function adds the forward and the residual edges to the graph.

int maximumFlow (int, int) – This function returns the max-flow value between the two nodes of a graph. **void print ()** - This function will display the final set of transactions.

bool bfs (int, int) - This function builds a level graph by using the *breadth-first search* traversal. It assigns level to each node and finds if there exists an augmenting path between source and sink.

int sendFlow (int, int, int, vector<int>&) – It finds the blocking flow by repeatedly calling the *depth-first search* traversal from source to the sink until the level graph is saturated. It augments the path from source to sink for each pass of the depth-first search. When the blocking flow is obtained, another level graph is built and the same procedure is followed until the graph is truly saturated.

set<tuple<string, string, int>> initialfn(vector<tuple<string, string, int>>)

Assign indices to names of creditors and debtors.

Add edges to graph.

Call maxflow() which will calculate maximum flow b/w each pair of vertices connected by an edge and printing the simplified transactions.

Void findAllUserTransactions()

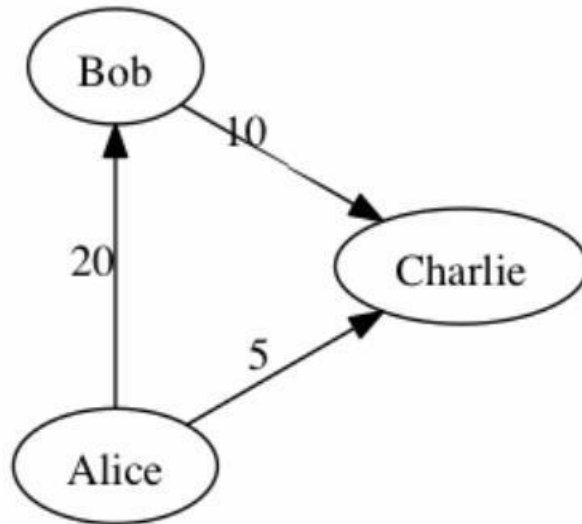
Find all the transactions the user is involved in

PROCEDURE

1. Feed the directed graph with given debts.
2. Select one of the non-visited edge from the directed graph.
3. Run Dinic's algorithm to determine the maximum flow of money possible between the two nodes of the selected edge. Also, compute the Residual Graph, which indicates the additional possible flow of debts between source and sink.
4. If maximum flow between source and sink is greater than zero, then add an edge with weight as max-flow to the Residual Graph.
5. Now go back to Step 1 and feed it the Residual Graph.
6. Once all the edges are visited, the Residual Graph obtained in the final iteration will contain the minimum number of transactions.

EXAMPLES

For example, if Alice owe Bob \$20 and Charlie \$5, Bob owe Charlie \$10, etc.



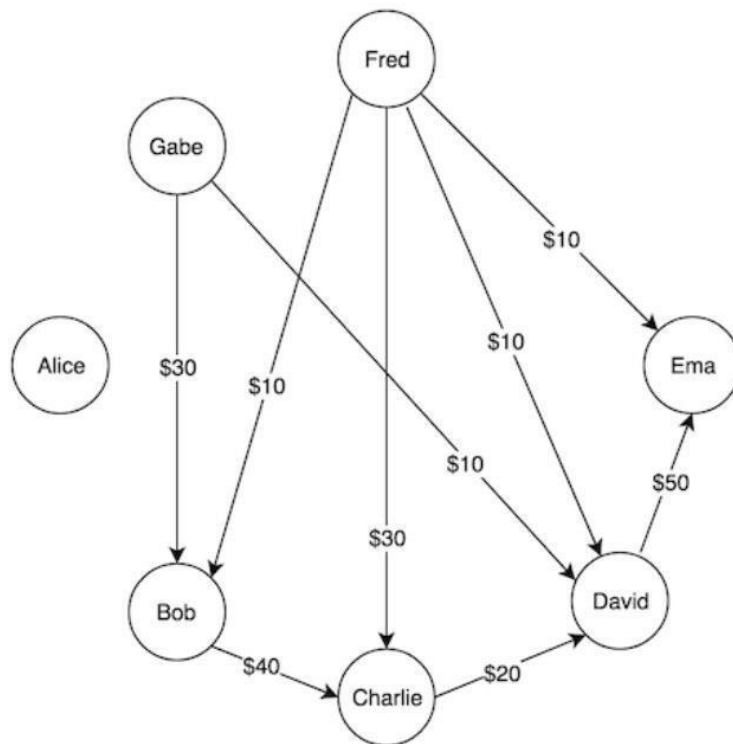
The above graph will be simplified to the following graph



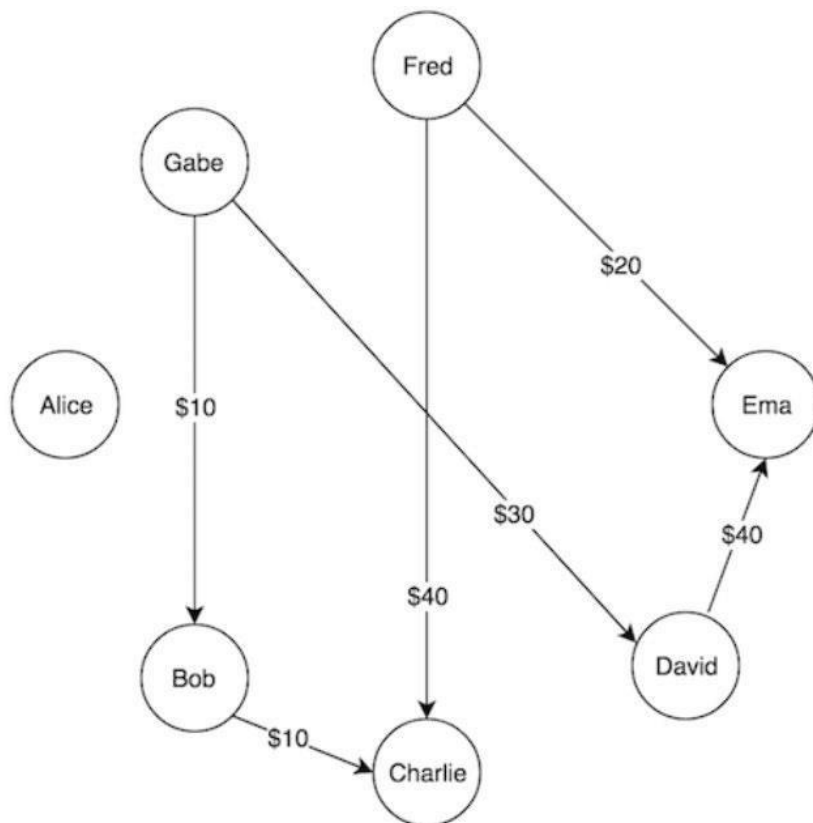
Also, we will make sure that the following rules are followed-

1. Everyone owes the same net amount at the end
2. No one owes a person that they did not owe before
3. No one owes more money in total than they did before the simplification

More complex network that can be simplified using the algorithm is shown below:




Representing Debts in the form of a Directed Graph



Simplified debts graph returned by the algorithm

SCREENSHOTS OF RUNNING CODE

LOGIN

 "D:\college\data struct lab\SPL_SEM_PROJECT\SPLITCOIN.exe"

```
Welcome to SPLITCOIN

Press 1 to login
Press 2 to signup
Press 3 to exit

Choose an option: 1
Enter username:
rosh
Welcome back rosh! These are your previous transactions:
clare owes dash $115
dash owes ken $3542
joe owes dash $3210
joe owes rosh $922
ken owes dash $150
lily owes rosh $7084
Would you like to delete these transactions? Press Y to delete and N to keep it
N
Given transactions

william owes clare $250
ken owes clare $200
clare owes dash $115
dash owes ken $3542
joe owes dash $3210
joe owes rosh $922
ken owes dash $150
lily owes rosh $7084

Final set of transactions after simplification:

dash owes ken $3542
joe owes dash $3210
joe owes rosh $922
ken owes clare $85
ken owes dash $265
lily owes rosh $7084
william owes clare $250
Maximum transaction: lily owes rosh amount 7084

Your transactions are:
joe owes rosh amount 922

lily owes rosh amount 7084

Process returned 0 (0x0)   execution time : 6.325 s
Press any key to continue.
```

SIGNUP

 "D:\college\data struct lab\SPL_SEM_PROJECT\SPLITCOIN.exe"

Welcome to SPLITCOIN

Press 1 to login

Press 2 to signup

Press 3 to exit

Choose an option: 2

Enter username:

rosh

Given transactions

joe owes clare \$245

joe owes rosh \$345

joe owes dash \$3542

clare owes rosh \$345

clare owes dash \$245

ken owes dash \$230

ken owes clare \$230

dash owes rosh \$542

rosh owes ken \$310

lily owes rosh \$3542

lily owes william \$3542

dash owes ken \$3542

william owes rosh \$3542

Final set of transactions after simplification:

clare owes dash \$115

dash owes ken \$3542

joe owes dash \$3210

joe owes rosh \$922

ken owes dash \$150

lily owes rosh \$7084

Maximum transaction: lily owes rosh amount 7084

Your transactions are:

joe owes rosh amount 922

lily owes rosh amount 7084

Process returned 0 (0x0) execution time : 4.189 s

Press any key to continue.

FUTURE SCOPE

1. It can be further modified to delete the transactions which have already been paid, making it more user friendly.
2. It can be turned into an app to make it available to a lot of people. Splitwise, a console app, has already been made, which had been our inspiration for this project.
3. It can also be modified such that big companies can use it to maintain their transactions.

REFERENCES

1. <https://medium.com/@mithunmk93/algorithm-behind-splitwises-debt-simplification-feature-8ac485e97688>
2. <https://www.youtube.com/watch?v=M6cm8UeeziI>

THANK YOU