Analysis of Algorithms

Proffesor: Andrés Méndez

Monday and thursday, 19:00hrs class

Nowadays, it is normal that people are connected between each other by any means possible. Social Networks such as Facebook and Twitter, between more, have proven that people are closer with others more than they know. But how do they keep track of how is a person connected with another? The purpose of this problem is to elaborate a time-efficient algorithm that can keep track of the relation between an user's belongings and the ones from another user. This, with the objective of a user having a faster interaction with the quality enhanced interface. As an application of this algorithm, the project will simulate the interaction of a user and his accounts as a member of a certain bank. As a member of this bank, the user can use his accounts to withdraw, deposit or transfer any money. When a user does this, a relation is created. This is where the algorithm boosts the way this relationships are kept. In such way that an account is used more often than any other, it is prioritized in such way that the user can access to it faster than before. Then again, when a transfer is done by a user to another user, the algorithm creates and keeps track of this movement for future references. In this way, it can enhance the quality of the interface that banks such as Santander, Bancomer and Banamex offers to the user. So, that it makes this app work faster for the user and this influences him for a more positive feedback to the bank.

System functional requierements

- The system needs to implement the insertion of different type of accounts (credit or debit) for a user.
- Insertion of new users that do not have an account.
- Insertion of cashiers inside an account in name of the Bank that counts as an user.
- Delete any kind of account such as a cashier or an user account.
- Search for a user's accounts and for a specific account.
- Insertion of relationships between accounts such as deposit, withdraw, transfer from a user account and deposit or withdraw from a cashier.
- Delete relationships between accounts.

The use of the algorithms

Through the elaboration of the project, the following Algorithms & Data Structures were implemented:

- DFS/BFS

  To traverse the nodes of the graph. If we are more interested in the closest first, we use BFS. If we are more interested in the furthest we can get, we use DFS.
- Edmond's (Analog of Prim's but for directed graphs)

  We can use Edmond's algorithm to get the spanning arborescence (minimum spanning tree) of the graph. This would mean, to use a unique relation to traverse all the nodes.
- Hashing

  Simple for the clients' id and the accounts' id. Necessary for insertion in both hash tables. The hash tables implemented will use linear probing.
- Heapsort

  To sort the accounts that belong to a client. Keeping the account that's been used the most as heap.
- Queue

  In order to keep track of the 15 most recent relations an account had. When a movement happens from an account into another, that movement is enqueued, so that a relation between both accounts involved is stored. If the queue is already full, the oldest movement is dequeued in order to enqueue the most recent one.

Complexities

With the description of the algorithms implemented in the software, the following complexities are given when running it:

Action	Average Case	Worst Case
Create New User	$\theta(1)$	$\mathcal{O}(U)$
Search User	$\theta(1)$	$\mathcal{O}(U)$
Delete User	$\theta(1)$	$\mathcal{O}(U)$
Create New Account	$\theta(1)$	$\mathcal{O}(U + A + log(A))$
Search Account	$\theta(1)$	$\mathcal{O}(A)$
Delete Account	$\theta(log(A))$	$\mathcal{O}(U + A + log(A))$
Make Transaction	$\theta(1)$	$\mathcal{O}(U + 2A + log(A))$
Search Transaction(DFS)	$\theta(A + T)$	$\mathcal{O}(2A + T)$
Search Transaction(BFS)	$\theta(A + T)$	$\mathcal{O}(2A + T)$
Edmonds's	$\theta(A + A^2 + T)$	$\mathcal{O}(A + A^2 + T)$

NOTE: A= Accounts U= Users T= Transactions (Most of the time T has a constant value of 15 as maximum, except when executing graph operations)

Functionality

First of all, a couple of assumptions were made in order to cover some functionalities of the software. One of them is that the Bank is an User, and Cashiers work as debit accounts of such Bank. In order to keep money flowing through the bank, the National Bank automatically gives more cash to the bank when it's balance reaches \$0. The cashiers are not related with other cashiers at all and they only receive relations; therefore, it will never be the origin of a relation, only the destiny. As mentioned before, only the 15 most recent relations between accounts are stored. The deposits and withdrawals from a Cashier are special. Deposit means that both the Account and the Cashier increase the balance amount. In addition, a Withdrawal means that the balance will decrease for both the Account and the Cashier. Finally, The relations between Owners and the Accounts are not stored in the adjacency table instead they are stored in the Hash Table and their corresponding trees.