Find actual activation date of phone numbers

Author: NguyenNguyen (nguyenbk92@gmail.com)

September 2018

1. Requirement analysis and design

a. Requirement analysis

- One phone number can occur multiple times in this list. So, we need to aggregate all transactions by phone number
- Input data is a messi stuff. We need to sort transactions of each phone number by activation date for easily process them

b. Datastructure

List of transaction. Each contains: phone number, activation date, deactivation date (can be empty)

```
[
    {
            Phone: '09800001',
            activation: '2018-09-01',
            deactivation: '2018-10-01'
    },
    {
            Phone: '09800001',
            activation: '2018-10-01',
            deactivation: '2018-11-01'
    }
]
```

c. Algorithm

- Validate data and bulk import to database
- Aggregate transactions by phone number
- Assume that the last activation (in transaction list of each phone number) is the actual activation date
- Traverse transaction list (from latest to oldest):
  - if the deactivation date of the previous transaction is more than 1 month older than the activation date of the current one, it means that we have an owner-transfer transaction. So the current one is first transaction of the latest owner, we should reassign the actual activation date by this activation date and stop
  - Otherwise, this is a renewal or update-plan transaction of the current owner, we need to continue to find the date they register this number

d. Pseudo code

- Read data from CSV file, validate and import to database (0)
- For each element (each phone number) (1)
    - Sort transaction list of each phone number (1a)
    - Assign actualActivationDate = last activation date in transaction list
    - i = length of transaction list – 1
    - Loop transaction list from the end to the beginning: while (i > 0) (1b)
        - if (transaction[i].activation - transaction[i-1].deactivation > 30 days)
            - actualActivationDate = transaction[i].activation
            - Export result for this number
            - exit the loop
        - else
            - i = i – 1
    - actualActivationDate = transaction[i].activation (// in this case, the activation date is defined by the first transaction)

e. Performance analysis

Given N is total records, K is average number of transactions regarding to one phone number (number of records regarding to one phone number)

$0 < K < N < 50\ 000\ 000$

- Read input data, validate, import data (0): the complexity is linear O(N)
- For each phone number: a linear loop with (N/K) phone number the complexity is O(N/K)
- Sort transaction list of each number (1a): square with K elements the complexity O(K) -> $O(K^2)$
- (1b): a linear loop with K transactions -> O(K)

The summary complexity:

$O(N + N/K * (K^2 + K)) = O(N +NK)$

Therefore:

- The worst case N = K: $O(N^2)$
- The best case: K = 1, the complexity is O(N)

2. Technology

a. Programing language

NodeJS

- Node.js is very efficient in managing asynchronous I/O from the root and is competent enough to solve the common web and network development problems. Besides the speedy JavaScript performance, the heart of Node.js is the Event Loop. To handle numerous clients, all I/O rigorous tasks in Node.js are undertaken together.

- JavaScript is present now in the browser as well as the server just because of Node.js.
- Easy to package dependencies with Node package management (npm)
- Easy for setting up unit tests

b. Database

Mongodb

- We need a database to store all transaction data and aggregate by phone number. It's okay to store in global variables but it's not a good solution if input is 50 million records
- Only one thing need to store, which is transaction record. So one simple database is needed. Mongodb is a simple one comparing to other relational databases
- Mongdb is easy to integrate with NodeJS and docker

3. Prototype, unit tests, Implementation, End-to-end tests

a. Prototype

- importData(): load input from CSV file
- processData(): parse raw data to our expected datastructure
- findActualActivationDate(): traverse transaction list of each number to find actual activation date
- exportData(): write output

b. Unit tests

- importData():I/O operation, no unit test
- validateData(): must have unit tests
  - Test empty record
  - Test invalid number
  - Test invalid date
  - Test missing deactivation date
  - Test a valid record
  - Test activation date is later deactivation date
- findActualActivationDate(): must have unit tests
  - Test empty record
  - Test missing record
  - Test number has only one owner
  - Test number has multiple owners
  - Test number has only one transaction
  - Test number has multiple transactions
- exportData(): I/O operation, no unit test

c. Implementation

- improtData(): use lib **fast-csv** for importing input data from CSV file
- This I/O operation is asynchonous, we need to make sure it's done before we do other operations
- validateData (): make sure
    - Phone number is number format
    - Activation date and deactivate must be date format YYYY-mm-dd
    - Activation date must be available
    - Deactivation date can be empty
    - For each record, deactivation date must be later than activation date
- findActualActivationDate(): implement this function following pseudo code segments
- exportData(): use lib **fast-csv**

d. End-to-end tests

Manual tests

4. Application delivery

- Fresh install and check dependencies and devDependencies
- Run unit tests again in new fresh invironment
- Sanity tests
- Complete documents

Docker: to eliminate all environment dependencies, dockerization is a good solution.
- Compose docker file
- Integrate Docker cloud CI to github repository
- Trigger automatically build new docker image whenever a new commit is pushed to this repository.

docker image: **nguyennguyen/hacker**

Usage: There are 2 docker need containers. One is our app container, another is mongodb
start with: **docker compose up**
Then **docker exec –it hacker bash**
And run: **node topics/find-actual-activation-date/script.js**