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 regarding to that 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

Parsing input data to a key-value object (*) that contains

- Key: phone number
- Value: transaction list of one phone number

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

c. Algorithm

- 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 (0)
- Process each row data (1)
    - Push row data to key-value object (*)
    - Sort the transaction list of that number by activation date (1a)

- For each element (each phone number) (2)
    - 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) (2a)
        - if (transaction[i].activation - transaction[i-1].deactivation > 1 month)
            - actualActivationDate = transaction[i].activation
            - exit the loop
        - else
            - i = i - 1
        - append actualActivationDate to result list
- Export result list to output file (3)

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)

$0 < K < N < 50000$

- Read input data (0): the complexity is linear O(N)
- Process each row data: O(N)
- Push new transaction and sort transaction list of the current number: because we sort this list again whenever we push new transaction to the list, the list should be almost sorted. The complexity is linear O(K)
- (2): a linear loop -> O(N)
- (2a): a linear loop -> O(K)
- (3) export data O(N)

The complexity is: $N + N*K + N*K + N = 2N (K + 1) = O(NK)$

Therefore:

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

No need. Data is simple. After we get input data, process it and immediately export output. There is no reason to make our application more complicated.

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

a. Prototpye

- 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
- processData(): must have unit tests
  - Test empty record
  - Test invalid number
  - Test invalid date
  - Test missing deactivation date
  - Test a valid record
- 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 **csvtojson** 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
- processData(): try to parse input raw data to our expected datastructure
- findActualActivationDate(): implement this function following pseudo code segments
- exportData(): use lib **json2csv**

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:

- **docker run –it nguyennguyen/hacker:latest**