1/ Strategy:

- Read line by line of CSV file and storing the key-value pair [phone_number - list_validity(activationDate, deactivationDate)] into memory

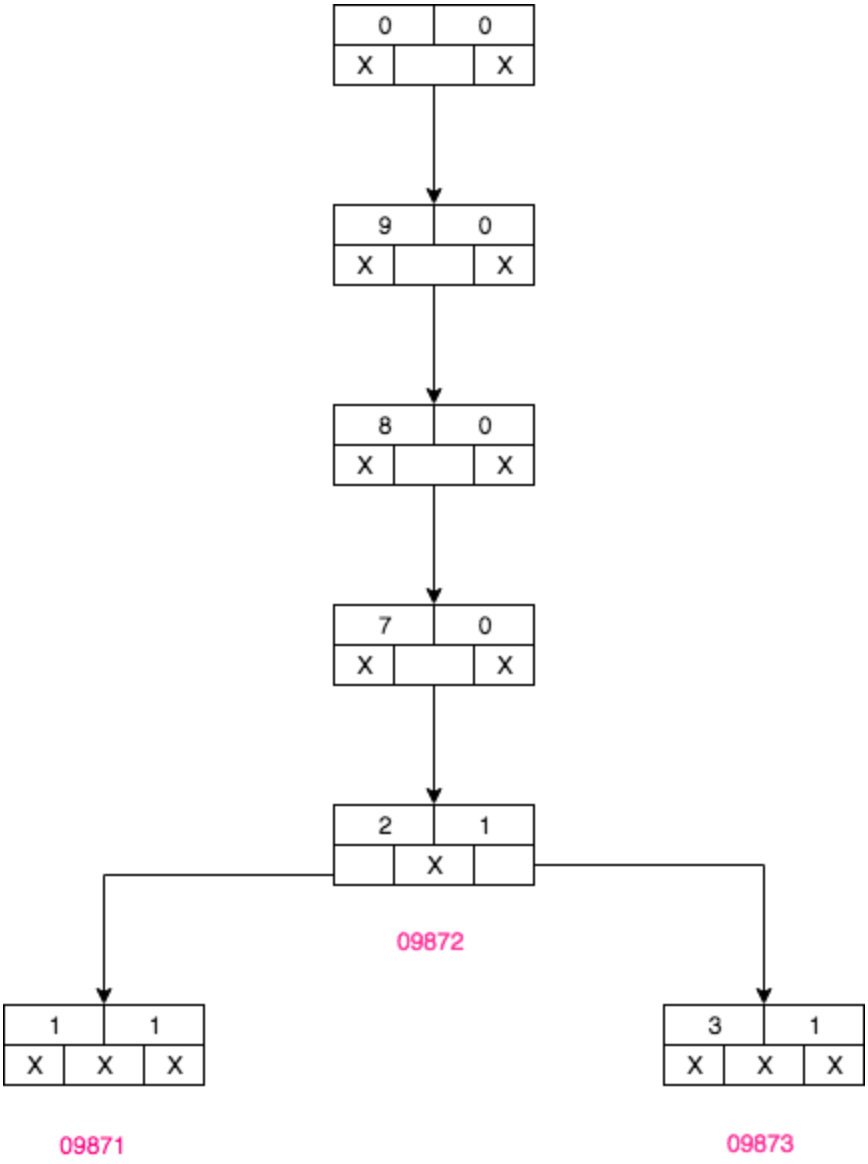
- Travervase via the map of such key-value pair to find the desired activation date in the list_validity of each phone_number

2/ Data structure:

a/ To store whole data as the key-value **[phone_number, list_validity]**, I choose **Ternary Tree** data structure which I can gain the optimized usage of space. The idea is to store each digit of a phone_number into the node and with that we can store some same nodes for several phone_numbers. For instance, if 2 phone_numbers contain the same prefix digits, they can share the spaces of memory to store those digits. Reference: https://en.wikipedia.org/wiki/Ternary_tree

- Each Node contains
  - **left_node**: the one having digit < the one of current node
  - **right_node**: the one having digit = the one of current node
  - **mid_node**: the root node of the sub tree where next digit of current node in phone_number is stored
  - **is_end**: a flag stating that the current node containing the last digit of a phone_number
  - **data**: containing the digit and list_validity not empty if the current node is_end=true.

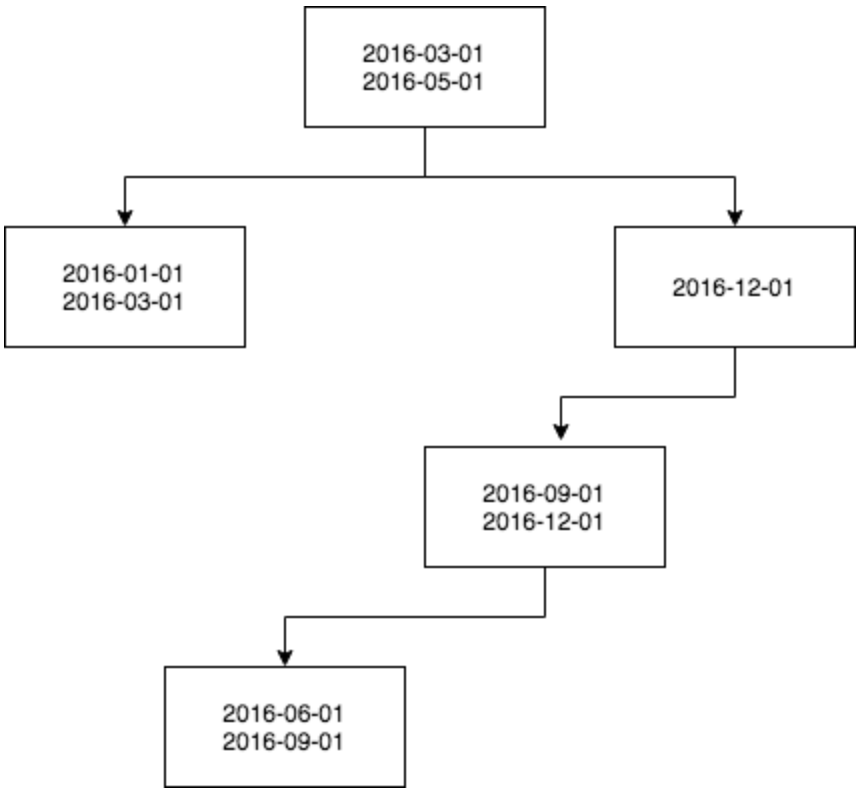
- The figure of nodes we store for 3 phone numbers **09871, 09872, 09873** could be



b/ To store the **list_validity,** I choose **Binary Sorted Tree** data structure. The idea is to take advantage of doing the sort during inserting storing the validity to memory.

- Each node contains:
  - **Left_node**: the node having the activation_date < the one of current node
  - **Right_node**: the node having the activation_date > the one of current node
  - **Data**: [activation_date, deactivate_date]

- The figure of data structure for 5 validities [2016-03-01, 2016-05-01], [2016-01-01, 2016-03-1], [2016-12-1,], [2016-09-01, 2016-12-01], [2016-06-01, 2016-09-01] could be



3/ Algorithm:

- a/ Traverse the **Ternary Tree** data structure to write the phone_number and its actual activation date by inorder order:
    - If node not exists, stop
    - Do traverse the left node with the string of previous digits without digit of current node
    - Append the digit of current node to the current string
    - If node is **is_end**, write the whole current string to file and actual activation date which is described in b)
    - Do traverse the right node with the string of previous digits without digit of current node
- b/ Traverse the **Binary Sorted Tree** to compute the actual activation date by inorder order:
    - Init the desired validity **result** = null
    - Do compute from the left
    - If current node not exists, do nothing
    - If **result** is not set, set **result** as the data of current node
    - Else if the **result deactivation_date** exists and < the **activation_date** of current node, set **result** = data of current node

- Otherwise, init **result** = [**activation_date** of interval, **deactivation_date** of current node]
- Do compute from the right.
- At the end, the **activation_date** of **result** is the desired value we want to print.

4/ Complexity

a/ Time:

- Given n = rows of file
- We need n times to read the whole file. Each row, we need insert the value to the **Ternary Tree** then it costs log(l*n) where l = length of a phone number. So we can consider it as log(n). Also, we need to add the validity to **list_validity (Binary Sorted Tree)** which will cost log(v) where **v is the number of validity of a phone number**. At this point, the cost of this process = n(log(n) + log(v)) ~ **nlog(n)** as v << n (1)
- Then, we do traverse **Ternary Tree** which will cost O(k) where **k = number of node in the tree**. During traverse the tree, we will do traverse the l**ist_validity(Binary Sorted Tree)** m (number of distinct phone number) times so the cost could be O(k +mv)) and it will be **O(n)** as k,v is small and m <= n (2)
- Eventually, from (1) and (2) we have complexity time of the algorithm = O(nlog(n)) + O(n) ~ **O(nlog(n))**

b/ Memory:

- Given n = rows of file
- I have no idea with the space to build the Ternary Tree but accoridng to https://en.wikipedia.org/wiki/String_searching_algorithm, the space is smaller than n
- We have to store the all the validities which will occupy **2 * n**
- Eventually the space for the algorithm could be **O(n)**

Summary: Time complexity: **O(nlog(n))**
Space complexity: **O(n)**