# Mobile phone tracking system

I have designed a data structure that will help us solve a simplified version of the mobile phone tracking problem, i.e., the fundamental problem of cellular networks: When a phone is called, find where it is located so that a connection may be established.

## Mobile phone tracking system description

As we know each mobile phone that is switched on is connected to the *base station* which is nearest. These base stations are popularly called cell phone towers. Although sometimes we may be within range of more than one base station, each phone is registered to exactly one base station at any point of time. When the phone moves from the area of one base station to another, it will be de-registered at its current base station and re-registered at new base station.

**Making a phone call.** When a phone call is made from phone $p_1$ registered with base station $b_1$ to a phone $p_2$, the first thing that the base station $b_1$ has to do is to find the base station with which $p_2$ is registered. For this purpose there is an elaborate technology framework that has been developed over time. You can read more about it on the Web. But, for now, we will assume that $b_1$ sends a query to a central server $C$ which maintains a data structure that can answer the query and return the name of the base station, let's call it $b_2$, with which $p_2$ is registered. $C$ will also send some routing information to $b_1$ so that $b_1$ can initiate a call with $b_2$ and, through the base stations $p_1$ and $p_2$ can talk. It is the data structure at $C$ that we will be implementing in this assignment.

**A hierarchical call routing structure.** We will assume that geography is partitioned in a hierarchical way. At the lowest level is the individual base station which defines an area around it such that all phones in that area are registered with it, e.g., all phones that are currently located in Bharti building,

School of IT and IIT Hospital are registered with the base station in Jia Sarai. This base station also serves phone in Jia Sarai and maybe some phones on Outer Ring Road in front of Jia Sarai. Further we assume that base stations are grouped into geographical locations served by an *level 1 area exchange*. So, for example, the Jia Sarai base station may be served by the Hauz Khas level 1 area exchange. Each level $i$ area exchange is served by a level $i$+1 area exchange which serves a number of level $i$ area exchanges, e.g., the Hauz Khas level 1 area exchange and the Malviya level 1 area exchange may be both served by a South-Central Delhi level 2 area exchange. A base station can be considered to be a level 0 area exchange in this hierarchical structure. Given a level $i$ exchange $f$, we say that the level $i$ + 1 exchange that serves it is the parent of $f$, and denote this parent($f$).

We will call this hierarchical call routing structure the *routing map* of the mobile phone network.

**Maintaining the location of mobile phones.** Every level $i$ area exchange, $e$, maintains a set of mobile phones, $S_e$, as follows. The set $S_e$ is called the *resident set* of $e$. The level 0 area exchanges (base stations) maintain the set of mobile phones registered directly with them. A level $i$+1 area exchange $e$, maintains the set $S_e$ defined as follows

$$s_e = [\, S_f,$$
$$\text{parent}(f)=e$$

i.e., the union of the sets of mobile phones maintained by all the level $i$ area exchanges it serves.

Clearly, the root of the routing map maintains the set of all currently registered mobile phones.

**Tracking a mobile phone.** The routing map along with the resident sets of each area exchange makes up the mobile phone tracking data structure we will be using. This data structure will be stored at the central server $C$. The process of tracking goes as follows.

- When a base station $b$ receives a call for a mobile phone with number $m$ it sends this query to $C$.
- If the root of the routing map is $r$, we first check if $m \in S_r$. If not then we tell $b$ that the number $m$ is "not reachable."
- If $m \in S_r$ we find that $e$ such that parent($e$) = $r$ and $m \in S_e$, i.e. we find the child of $r$ which contains $m$ in its resident set.
- Continue like this till we reach all the way down to a leaf of the routing map. This leaf is a base station $b^0$. The central server sends $b^0$ to $b$ along with the path in the routing map from $b$ to $b^0$.

I now describe the implementation details of the system defined above. This is split into two programming projects, #1 and #2

## Project 1

- A java class called Myset that implements sets, with the following methods:
  - public Boolean IsEmpty(): returns true if the set is empty.
  - public Boolean IsMember(Object o): Returns true if o is in the set, false otherwise.
  - public void Insert(Object o): Inserts o into the set.
  - public void Delete(Object o): Deletes o from the set, throws exception if o is not in the set.
  - public Myset Union(Myset a): Returns a set which is the union of the current set with the set a.
  - public Myset Intersection(Myset a): Returns a set which is the intersection of the current set with the set a.

    **Optional challenge problem:** *Bloom filters* are an efficient data structure for implementing the class Myset. Yet to work on !!

- A java class called MobilePhone, with the following methods:
  - MobilePhone(Int number): constructor to create a mobile phone. Unique identifier for a mobile phone is an integer.
  - public Int number(): returns the id of the mobile phone.
  - public Boolean status(): returns the status of the phone, i.e. switched on or switched off.
  - public void switchOn(): Changes the status to switched on.
  - public void switchOff(): Changes the status to switched off.
  - public Exchange location(): returns the base station with which the phone is registered if the phone is switched on and an exception if the phone is off. The class Exchange will be described next.
- A class MobilePhoneSet which stores MobilePhone objects in a Myset.

•

A class ExchangeList which implements a linked list of exchanges.

- A java class Exchange that will form the nodes of the routing map structure.
  - Exchange(Int number): constructor to create an exchange. Unique identifier for an exchange is an integer.
  - All usual Node methods for a general tree like public Exchange parent(), public Exchange numChildren() (for number of children), public Exchange child(int i) (returns the *i*th child), public Boolean isRoot(), public RoutingMapTree subtree(int i) (returns the *i*th subtree) and any other tree methods you need. The class definition RoutingMapTree will be defined later.
  - public MobilePhoneSet residentSet(): This returns the resident set of mobile phones of the exchange.
- A java class RoutingMapTree which is a tree class whose nodes are from the Exchange class:
  - RoutingMapTree(): constructor method. This should create a RoutingMapTree with one Exchange node, *the root node which has an identifier of 0.* Create other constructors that you deem necessary.
  - All general tree methods like *public Boolean containsNode(Exchange a)* but with Exchange as the node class.
  - public void switchOn(MobilePhone a, Exchange b): This method only works on mobile phones that are currently switched off. It switches the phone a on and registers it with base station b. The entire routing map tree will be updated accordingly.
  - public void switchOff(MobilePhone a): This method only works on mobile phones that are currently switched on. It switches the phone a off. The entire routing map tree has to be updated accordingly.

- public void performAction(String actionMessage): This the main stub method that you have to implement. It takes an action as a string. The list of actions, and their format will be described next.

The primary task is to give the correct answer to the *query* messages (described later). We will verify the output of the program during the demo. Here is a list of action messages that you need to handle:

- addExchange a b This should create a new Exchange b, and add it to the child list of Exchange a. If node a has n children, b should be its $(n+1)^{th}$ child. If there is no Exchange with identifier a, then throw an Exception.
- switchOnMobile a b This should switch ON the mobile phone a at Exchange b. If the mobile did not exist earlier, create a new mobile phone with identifier a. If there is no Exchange with an identifier b, throw an exception.
- switchOffMobile a This should switch OFF the mobile phone a. If there is no mobile phone with identifier a, then throw an Exception.
- queryNthChild a b This should print the identifier of the Exchange which is the (b)th child of Exchange a.
- queryMobilePhoneSet a This should print the identifier of all the mobile phones which are part of the resident set of the Exchange with identifier a.

Now, move on to implementing the details of the tracking system. Implement the following additional methods in the RoutingMapTree class.

- public Exchange findPhone(MobilePhone m): Given a mobile phone m it returns the level 0 area exchange with which it is registered or throws an exception if the phone is not found or switched off.
- public Exchange lowestRouter(Exchange a, Exchange b): Given two level 0 area exchanges a and b this method returns the level *i* exchange with the smallest possible value of *i* which contains both a and b in its subtree. If a = b then the answer is a itself.

- 

- public ExchangeList routeCall(MobilePhone a, MobilePhone b): This method helps initiate a call *from* phone a *to* phone b. It returns a list of exchanges. This list starts from the base station where a is registered and ends at the base station where b is registered and represents the shortest route in the routing map tree between the two base stations. It goes up from the initiating base station all the way to the lowestRouter connecting the initiating base station to the final base station and then down again. The method throws exceptions as appropriate.

- public void movePhone(MobilePhone a, Exchange b): This method modifies the routing map by changing the location of mobile phone a from its current location to the base station b. Note that b must be a base station and that this operation is only valid for mobile phones that are currently switched on.

Here is a list of the extra action messages that RoutingMapTree support.

- queryFindPhone a This should print the identifier of the exchange returned by the findPhone(MobilePhone m) method. Here, m represents the mobile phone whose identifier is a.
- queryLowestRouter a b This should print the identifier of the exchange returned by the lowestRouter(Exchange e1, Exchange e2) method. Here, e1 and e2 represent exchanges with identifier a and b respectively.
- queryFindCallPath a b This should print the list returned by the routeCall(MobilePhone m1, MobilePhone m2) method. Here, m1 and m2 represent mobile phones with identifier a and b respectively. Successive entries in the list should be separated by a comma.
- movePhone a b This action should set the level 0 area exchange of the mobile phone with identifier a to exchange with identifier b. Throw exception if mobile a is not available, or exchange b does not exist.

make this whole thing much closer to reality by writing a multithreaded implementation of the entire system. Yet to do !!