```
// Copy paste this Java Template and save it as "PatientNames.java"
import java.util.*;
import java.io.*;

// write your matric number here: A0154973U
// write your name here: Ahmad Syafiq
// write list of collaborators here: Muhammad Afiq Lattif, Syahiran Rafi
// year 2017 hash code: 7TVOcaRb0L0GfdsoDnh5 (do NOT delete this line)
class PatientNames {
  // if needed, declare a private data structure here that
  // is accessible to all methods in this class

  // -----------------------------------------
        public BBST malePatients; //tree for male patients
        public BBST femalePatients; //tree for female
        public BBST bothPatients;       //combined tree

  // -----------------------------------------

  public PatientNames() {
    // Write necessary code during construction;
    //
    // write your answer here

    // -----------------------------------------
          malePatients = new BBST();
          femalePatients = new BBST();
          bothPatients = new BBST();

    // -----------------------------------------
  }

  void AddPatient(String patientName, int gender) {
    // You have to insert the information (patientName, gender)
    // into your chosen data structure
    //
    // write your answer here

    // -----------------------------------------
          Patient patient = new Patient(patientName, gender);

            switch(gender){
              case 0:
                malePatients.insert(patient);
                femalePatients.insert(patient);
                bothPatients.insert(patient);
                break;
              case 1:
                malePatients.insert(patient);
                bothPatients.insert(patient);
                break;
              case 2:
                femalePatients.insert(patient);
                bothPatients.insert(patient);
                break;
              default:
                break;
            }

    // -----------------------------------------
  }

  void RemovePatient(String patientName) {
    // You have to remove the patientName from your chosen data structure
```

```
    //
    // write your answer here

    // ------------------------------------------
          malePatients.delete(patientName);
          femalePatients.delete(patientName);
      bothPatients.delete(patientName);
    // ------------------------------------------
  }

  int Query(String START, String END, int gender) {
    int ans = 0;

    // You have to answer how many patient name starts
    // with prefix that is inside query interval [START..END)
    //
    // write your answer here

    // ------------------------------------------
    if(gender == 0)
        ans= bothPatients.countSubTree(START, END);
      else if(gender == 1)
        ans= malePatients.countSubTree(START, END);
      else
        ans= femalePatients.countSubTree(START, END);


    // ------------------------------------------

    return ans;
  }

  void run() throws Exception {
    // do not alter this method to avoid unnecessary errors with the automated judging
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter pr = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(System.out)));
    while (true) {
      StringTokenizer st = new StringTokenizer(br.readLine());
      int command = Integer.parseInt(st.nextToken());
      if (command == 0) // end of input
        break;
      else if (command == 1) // AddPatient
        AddPatient(st.nextToken(), Integer.parseInt(st.nextToken()));
      else if (command == 2) // RemovePatient
        RemovePatient(st.nextToken());
      else // if (command == 3) // Query
        pr.println(Query(st.nextToken(), // START
                         st.nextToken(), // END
                         Integer.parseInt(st.nextToken()))); // GENDER
    }
    pr.close();
  }

  public static void main(String[] args) throws Exception {
    // do not alter this method to avoid unnecessary errors with the automated judging
    PatientNames ps2 = new PatientNames();
    ps2.run();
  }
}

class Patient {
  public int gender;
  public String name;
```

```
  public Patient(String name, int gender){
    this.name=name;
    this.gender=gender;
  }
}

class Vertex {
  public Patient patient;
  public Vertex parent,left,right;
  public int height,size;

  public Vertex(Patient patient){ //when vertex created, parent and child are null. height
0 and size 1.
    this.patient = patient;
    parent = null;
    left = null;
    right = null;
    height=0;
    size = 1;
  }
}

class BBST {
  protected Vertex root;

  public BBST(){
    root=null;
  }

  public int getSize(Vertex vertex){ //returns size of vertex. if null return 0
    if(vertex == null) return 0;
    return vertex.size;
  }
  public int getHeight(Vertex vertex){ //returns height of vertex. if null return -1
          if(vertex ==null) return -1;
          return vertex.height;
  }
  public int findRank(Vertex vertex, String key){ //finds rank of vertex.
    int rank = 0;
    while(vertex!=null){ //traverse tree by comparing while keeping track of rank
      if(key.compareTo(vertex.patient.name)<0) //if smaller go left
          vertex= vertex.left;
      else if (key.compareTo(vertex.patient.name)>0){ //if larger go right
          rank+= getSize(vertex.left) +1; //get size of left subtree +1 and add to rank.
(since left subtree and current vertex are all less)
          vertex = vertex.right;
      }
      else return rank + getSize(vertex.left); //if equal return rank and leftsubtree
size.
    }
    return -1; //if not found
  }
  public int balanceFactor(Vertex vertex){ //computes balance factor by subtracting right
height from left height and returns it
          return getHeight(vertex.left) - getHeight(vertex.right);

  }
  public Vertex update(Vertex vertex){ //updates vertex height and size and returns it
          vertex.height=Math.max(getHeight(vertex.left), getHeight(vertex.right))+1;
          vertex.size=getSize(vertex.left) + getSize(vertex.right)+1;
          return vertex;
  }
  public Vertex rotateLeft(Vertex vertex){
          Vertex rightChild = vertex.right;
          rightChild.parent = vertex.parent;
```

```
          vertex.parent = rightChild;
          vertex.right = rightChild.left; //make left child of right child right child of
vertex
          if(rightChild.left!=null) rightChild.left.parent= vertex;
          rightChild.left = vertex; //make vertex left child of right child
          vertex=update(vertex); //update vertex attr
          rightChild=update(rightChild); //update rightChild attr
          return rightChild; //return rightChild(the new parent)
  }
  public Vertex rotateRight(Vertex vertex){
          Vertex leftChild = vertex.left;
          leftChild.parent = vertex.parent;
          vertex.parent = leftChild;
          vertex.left = leftChild.right;
          if(leftChild.right!=null) leftChild.right.parent= vertex;
          leftChild.right = vertex;
          vertex=update(vertex);
          leftChild=update(leftChild);
          return leftChild;
  }

public Vertex balance(Vertex vertex){
          int bf = balanceFactor(vertex); //get balanceFactor
//        System.out.println(vertex.patient.name+bf);
//        if(vertex==null) return vertex;
          if(bf>1){ //if bf 2
                  if(balanceFactor(vertex.left)<0){  //if bf of leftchild -1
                          vertex.left=rotateLeft(vertex.left);
                          vertex=rotateRight(vertex);
                  }
                  else vertex=rotateRight(vertex);
          } else if (bf<-1){ //if bf -2
                  if(balanceFactor(vertex.right)>0){ //if bf of right child 1
                          vertex.right=rotateRight(vertex.right);
                          vertex=rotateLeft(vertex);
                  }
                  else vertex=rotateLeft(vertex);
          }
          return vertex;
  }
  public void insert(Patient patient) {

            root = insert(root, patient);

          }
  public Vertex insert(Vertex vertex, Patient patient){
          if(vertex == null){ //once reach null,create vertex and return it
                  return new Vertex(patient);
          }
          if(patient.name.compareTo(vertex.patient.name)>0){ //name larger than current
vertex, go right
                  vertex.right = insert(vertex.right, patient);
                  vertex.right.parent = vertex;
          }
          else{
                  vertex.left=insert(vertex.left, patient);
                  vertex.left.parent = vertex;
          }
          vertex=update(vertex);
          vertex = balance(vertex);
          return vertex;
  }
```

```
  public Vertex findMin(Vertex vertex){
          if (vertex != null)
            if (vertex.left == null) return vertex;
            else return findMin(vertex.left);
          else return null;
  }
  public Vertex findMax(Vertex vertex){
          if (vertex != null)
                    if (vertex.right == null) return vertex;
                    else return findMax(vertex.right);
                  else return null;
  }
  public Vertex successor(Vertex vertex){ //find successor of vertex
          if(vertex.right == null){ //if cant go right
                  Vertex parent = vertex.parent;
                  Vertex temp = vertex;
                  while((parent!=null)&&(temp == parent.right)){ //while can go up and its
a right child(the parent is smaller)
                          temp = parent;
                          parent = temp.parent;
                  }
                  return parent;
          }
          else return findMin(vertex.right);
  }
  public Vertex predecessor(Vertex vertex){
          if(vertex.left == null){
                  Vertex parent = vertex.parent;
                  Vertex temp = vertex;
                  while((parent!=null)&&(temp == parent.left)){
                          temp = parent;
                          parent = temp.parent;
                  }
                  return parent;
          }
          else return findMax(vertex.left);
  }
  public Vertex find(Vertex vertex, String key){ //finds vertex closest to given key.
          if(key.equals(vertex.patient.name)) return vertex;
          else if (key.compareTo(vertex.patient.name)<0){ //if smaller
                  if(vertex.left!=null)
                          return find(vertex.left, key); //go left if not null
                  else //else return current vertex
                          return vertex;
          } else {
                  if(vertex.right!=null)
                          return find(vertex.right, key);
                  else
                          return vertex;
          }
  }
  public void delete(String name){ //calls delete name start from root
          root = delete(root,name);
  }
  public Vertex delete(Vertex vertex, String key){
          if(vertex == null) return null; //not found return null
          if(vertex.patient.name.equals(key)){ //if found
                  if(vertex.left == null && vertex.right ==null) //if no child set it as
null
                          vertex=null;
                  else if(vertex.left == null && vertex.right !=null){ //if have only
right child replace it
                          vertex.right.parent = vertex.parent;
                          vertex = vertex.right;
                  }
```

```
                else if(vertex.left !=null && vertex.right == null){
                        vertex.left.parent = vertex.parent;
                        vertex = vertex.left;
                }
                else { //if have right and left child. find successor and replace it.
then delete successor
                        Vertex successor = successor(vertex);
                        vertex.patient=successor.patient;
                        vertex.right = delete(vertex.right,successor.patient.name);
                }
        }
        else if(key.compareTo(vertex.patient.name)>0){
                vertex.right = delete(vertex.right, key);
        }
        else{
                vertex.left = delete(vertex.left, key);
        }
        if(vertex!=null){
                vertex=update(vertex);
                vertex = balance(vertex);
        }
        return vertex;
  }
  public int countSubTree(String start, String end){ //finds number of vertex in given
range
        if(root == null) return 0; //if empty return 0
        Vertex first = find(root,start); //find name closest to start
        Vertex last = find(root,end); //find name closest to end
        int firstRank = findRank(root,first.patient.name); //find their rank
        int lastRank = findRank(root, last.patient.name);
        int sum = lastRank-firstRank+1; //get number of vertices in between
        if(start.compareTo(first.patient.name)>0) //if first name is smaller than start,
remove it
                sum--;
        if(end.compareTo(last.patient.name)<=0) //if last name is equal or larger than
end. remove it
                sum--;
        if(first.equals(last)) //if first and last are the same there is only one
element. return 1 or 0 if name was removed above
        {
                if(sum>0)
                        return 1;
                else
                        return 0;
        }
                return sum;
  }
}
```