```
/* Problem 2--Roman Piazzas
   What I did was sort both lists of entries and use binary search to
   find connecting rows and columns. */

import java.io.*;
import java.util.*;

public class prob2 {

 private static Scanner in;
 private static PrintWriter out;
 private static int cs;
 private static int N, P, Q, mnz, nz;
 private static int[][] M1, M2, M3;

 public static void main (String[] args) throws Exception {

  in = new Scanner (new File ("prob2.in"));
  out = new PrintWriter ("prob2.out");
  cs = 1;
  while (true) {
   N = in.nextInt(); //size of matrix.  I don't need it
   if (N==0) break;  //so I throw it away.
   ReadIn(); //Read in data
   Process (); //Process data
   out.printf (
    "Case %d: The number of nonzero entries is %d and the maximum "+
    "nonzero entry is %d.\r\n\r\n",cs++,nz,mnz);
  }
  in.close ();
  out.close ();
 }

 //Process computes the matrix product

 public static void Process () {

  mnz = 0;  nz = 0;
  int m3sz = 0; //size of unsummed product matrix
  for (int i=0; i < P; i++) {
   int[] rng2 = SearchMat (M2,M1[i][1]); //Binary search second matrix
   m3sz += rng2[1]-rng2[0]+1; //for appropriate rows and add to count
  }
  M3 = new int[m3sz][3]; //Allocate space for product
  m3sz = 0;
  for (int i=0; i < P; i++) { //For each row in first matrix
   int[] rng2 = SearchMat (M2,M1[i][1]); //Binary search second
   for (int j = rng2[0]; j <= rng2[1]; j++) {  //For appropriate rows
    int[] entry = new int[3];  //Compute product and add to new matrix
    entry[0] = M1[i][0];
    entry[1] = M2[j][1];
    entry[2] = M1[i][2]*M2[j][2];
    M3[m3sz++] = entry;
   }
  }
  SortMat (M3); //Sort new matrix so identical positions are adjacent
  int i = 0;
  while (i < m3sz) { //Sum matching row and column positions
   int j = i+1;
   while (j < m3sz && M3[i][0]==M3[j][0] && M3[i][1]==M3[j][1]) {
    M3[i][2] += M3[j][2];
    j++;
   }
   if (M3[i][2] != 0) { //Check for nonzero, keeping track of max
```

```
 65      nz++;
 66      if (mnz==0 || M3[i][2] > mnz) mnz = M3[i][2];
 67     }
 68     i = j;
 69    }
 70   }
 71
 72   //ReadIn reads in the matrices.
 73   public static void ReadIn () {
 74
 75    P = in.nextInt();  //Size of first
 76    M1 = new int[P][3];  //Individual positions of first
 77    for (int i=0; i < P; i++)
 78     for (int j=0; j < 3; j++)
 79      M1[i][j] = in.nextInt();
 80    Q = in.nextInt();  //Size of second
 81    M2 = new int[Q][3];  //Individual positions of second
 82    for (int i=0; i < Q; i++)
 83     for (int j=0; j < 3; j++)
 84      M2[i][j] = in.nextInt();
 85    SortMat (M1);  //Sort matrices
 86    SortMat (M2);
 87   }
 88
 89   //SortMat uses a simple Shell Sort to sort first by row then by
 90   //column
 91   public static void SortMat (int[][] M) {
 92
 93    int sz = M.length;
 94    for (int d=sz-1; d >= 1; d--) {
 95     int dd = d;
 96     while (dd%2==0) dd/=2;
 97     while (dd%3==0) dd/=3;
 98     if (dd==1)
 99      for (int i=d; i < sz; i++)
100       if (M[i-d][0] > M[i][0] ||
101           M[i-d][0]==M[i][0] && M[i-d][1] > M[i][1]) {
102        int[] t = M[i-d];
103        M[i-d] = M[i];
104        M[i] = t;
105       }
106    }
107   }
108
109   //Binary searches a matrix for a range representing the rows matching r
110   public static int[] SearchMat (int[][] M, int r) {
111
112    int sz = M.length;
113    int f = 0, l=sz-1, m=0;
114    int[] ret = new int[2];
115    boolean found = false;
116    while (f <= l && !found) {
117     m = (f+l)/2;
118     found = M[m][0] == r;
119     if (!found)
120      if (M[m][0] < r) f = m+1;
121      else l = m-1;
122    }
123    if (!found) {ret[0]=f; ret[1]=l;}
124    else {
125     int ff=f, ll=m-1;
126     while (ff <= ll) {
127      int mm = (ff+ll)/2;
128      if (M[mm][0]==r) ll=mm-1;
```

```
     else ff=mm+1;
    }
    ret[0]=ff;
    ff=m+1; ll=l;
    while (ff <= ll) {
     int mm = (ff+ll)/2;
     if (M[mm][0]==r) ff=mm+1;
     else ll=mm-1;
    }
    ret[1]=ll;
   }
   return ret;
  }
 }
```