Marathon Match

Problem Statement

Contest: MM 2

Problem: AsteroidDetector

Problem Statement

Prizes

The best 5 performers of this contest (according to system test results) will receive the following prizes:

1. place: 8000$
2. place: 5500$
3. place: 3500$
4. place: 2000$
5. place: 1000$

Background

Life as we know it may very well change at any moment for ever without us even realizing it... but don't despair, NASA and Planetary Resources have spun up their engines and they have a plan. This time your task is to develop an algorithm that can detect asteroids and Near Earth Objects (NEOs) from a sequence of images.

Your algorithm will received the following input:

- The raw image data from 4 (FITS) images of the sky, taken roughly 10 minutes apart. The resolution of each set of 4 images will be the same, but the number of pixels, pixel pitch, and noise between sets may vary. The data contains 16 bit values.
- The FITS header information for each of the FITS images.
- A detection list associated with the set of images which contains a list of information for known detected objects. This will only be provided to your algorithm during the training phase.

The training data can be downloaded here. The image file is in raw 16 bit format and can be read in and displayed by the visualizer that can be downloaded here. The SAOImage DS9 tool can be used to view and inspect FITS images, the software can be downloaded here. The FITS images were compressed with the hcompress utility which can be used to decompress the images in order to view them in the DS9 tool.

Implementation

Your task is to implement a `trainingData`, `testingData` and `getAnswer` methods, whose signatures are detailed in the Definition section below.

`imageData_1` contains the image data for frame 1, `imageData_2` for frame 2, etc. The array contains image data of size `width` * `height`. Every element will contain a 16 bit value which ranges roughly from 0 to 65535, some of the values might go slightly out of this range. Let (x,y) be the coordinate within an image. (0,0) will be the top left corner in the image and (width, height) the bottom right corner. The pixel data can then be found at index `[x + y*width]` of the `imageData_X` arrays.

`header_1, .. header_4` contains the FITS header of the images. Each element will contain a row in the header.

`wcs_1, .. wcs_4` contains the data extracted from the FITS headers in order to convert (x,y) coordinates to (Right Ascension, Declination) World Coordinate System (WCS) coordinates and vice-versa. Each of the arrays contain 8 double values. Please see the java source code in the visualizer that can perform the coordinate conversions. Specifically, the `convertRADEC2XY` and `convertXY2RADEC` methods in the visualizer source code perform the conversions. More information on the WCS is available here and here.

`detections` contain information about the known objects. The detection list is in space delimited format with 8 columns and each row representing a single detection in one of the 4 FITS images. The columns are:

columns and each row representing a single detection in one of the 4 FITS images. The columns are:

1. Unique ID - An identifier for what detected object a row belongs to.
2. Frame Number - which observation is this row relevant to (1, 2, 3 or 4)
3. RA - right ascension of object in decimal hours
4. DEC - declination in decimal degrees
5. X - location in pixels of the object in the FITS image.
6. Y - location in pixels of the object in the FITS image.
7. Magnitude - brightness of the object in magnitudes.
8. Neo - this value will be 1 if the object is a Near Earth Object (NEO), 0 otherwise.

`imageID` provides an unique identifier for each testing image set.

Firstly, your `trainingData` method will be called with FITS images, headers and known detected objects. Your method will be called 100 times, one time for every set in the training data. You can use this method to train your algorithm on the provided data if you want to. If your `trainingData` method returns the value 1, no more training data will be passed to your algorithm and the testing phase will begin.

Secondly, your `testingData` method will be called 20 times with different image data than provided during training. The `testingData` method can return any value, it does not matter what you return.

Finally, your `getAnswer` method will be called. This method should return a list of all the objects that your algorithm detected for each set provided to your algorithm through the `testingData` method. Your may not return more than 100000 detections. Each element in your return should contain the following information in space delimited format:

1. imageID - the imageID associated with the image where the object was detected
2. RA_1 - right ascension of object in decimal hours in frame 1.
3. DEC_1 - declination in decimal degrees in frame 1.
4. RA_2 - right ascension of object in decimal hours in frame 2.
5. DEC_2 - declination in decimal degrees in frame 2.
6. RA_3 - right ascension of object in decimal hours in frame 3.
7. DEC_3 - declination in decimal degrees in frame 3.
8. RA_4 - right ascension of object in decimal hours in frame 4.
9. DEC_4 - declination in decimal degrees in frame 4.
10. NEO - this value should be 1 if your algorithm think the object is a Near Earth Object (NEO), 0 otherwise.

The goal is to order those elements (detections) in such a way, that those that you believe are the **most** probable to be objects and NEO's at the **front** of the array, and those the **least** probable at the **back**.

Testing and scoring

There are 1 example, 10 provisional tests and at least 20 system tests. Each test will contain data from 20 image sets for testing and 100 image sets for training.

Scoring is calculated based on average precision for each test case as follows:

Suppose that known detections for the given test case consists of A detections C(0), C(1), ..., C(A-1) and your solution returned B detections D(0), D(1), ..., D(B-1), in this exact order. Let's say that two detections match if the sum of their squared distance between their RA and DEC locations on all 4 frames is strictly less than **0.001**. A detection will be considered correct if the reported detection matches with at least one of the detections in the known detections list for the same image. A bonus score will be applied if the NEO field of the detection matches and the known object was marked as a NEO.

The following pseudo code will be used to calculate your score:

```
matched := array[0..A-1] of booleans, initialized with False values
score := 0.0
detected := 0
neo_count := 0
neo_detected := 0
for i := 0, 1, ..., B-1
{
```

```
        if (D(i) is NEO) neo_count := neo_count + 1
        for j := 0, 1, ..., A-1
        {
                if (matched[j] == False) and (C(j).imageID == D(i).imageID) and
                   (sum of squared distance between C(j) and D(i) < 0.001)
                {

                        matched[j] := True
                        detected := detected + 1
                        score := score + (1000000.0 / A) * (detected / (i + 1))
                        if (C(j) is NEO)
                        {
                                neo_detected := neo_detected + 1
                                score := score + (100000.0 / numOfNEOs) * (neo_detected / (neo_count+1))
                        }
                        Break
                 }
        }
}
```

You can see these scores for example test cases when you make example test submissions. If your solution fails to produce a proper return value, your score for this test case will be 0.

The overall score on a set of test cases is the arithmetic average of scores on single test cases from the set. The match standings displays overall scores on provisional tests for all competitors who have made at least 1 full test submission. The winners are competitors with the highest overall scores on system tests.

An offline tester/visualizer tool is available.

The current state of the art - Catalina algorithm - allows to detect 15-20% of the brightest objects from the list, what corresponds to a score between 100000-150000. Hope you beat it!

Special rules and conditions

- The allowed programming languages are C++, Java, C# and VB. Python submissions will not be accepted.
- You can include open source code in your submission. Open Source libraries under the BSD, or GPL or GPL2 license will be accepted. Other open source licenses could be accepted too. Just be sure to ask us.
- In order to receive the prize money, you will need to fully document your code and explain your algorithm. If any parameters were obtained from the training data set, you will also need to provide the program used to generate these parameters. There is no restriction on the programming language used to generate these training parameters. Note that all this documentation should not be submitted anywhere during the coding phase. Instead, if you win a prize, a TopCoder representative will contact you directly in order to collect this data.
- You may not use any external (outside of this competition) source of data to train your solution.

Definition

Class:	AsteroidDetector
Method:	trainingData
Parameters:	int, int, int[], String[], double[], int[], String[], double[], int[], String[], double[], int[], String[], double[], String[]
Returns:	int
Method signature:	int trainingData(int width, int height, int[] imageData_1, String[] header_1, double[] wcs_1, int[] imageData_2, String[] header_2, double[] wcs_2, int[] imageData_3, String[] header_3, double[] wcs_3, int[] imageData_4, String[] header_4, double[] wcs_4, String[] detections)
Method:	testingData
Parameters:	String, int, int, int[], String[], double[], int[], String[], double[], int[], String[], double[], int[], String[], double[]
Returns:	int
Method signature:	int testingData(String imageID, int width, int height, int[] imageData_1, String[] header_1, double[] wcs_1, int[] imageData_2, String[] header_2, double[] wcs_2, int[] imageData_3, String[] header_3, double[] wcs_3, int[] imageData_4, String[] header_4, double[] wcs_4)

Method: getAnswer
Parameters:
Returns: String[]
Method signature: String[] getAnswer()
(be sure your methods are public)

Notes

- The match forum is located here. Please check it regularly because some important clarifications and/or updates may be posted there. You can click "Watch Forum" if you would like to receive automatic notifications about all posted messages to your email.
- You can train your solution offline based on the given training data file and you can hardcode data into your solution. However remember that you can't use data from other sources than this contest.
- Memory limit is 4096MB. Time limit is 80 minutes per test case which includes only the time spent in your code. Solutions are executed at VMs. Therefore the amount of available CPU resources may vary to some degree. The time limit is set to a large value in order to help you deal with that. Given this variability, we recommend you to design your solution so that its estimated runtime does not exceed 60 minutes.
- There is no explicit code size limit. The implicit source code size limit is around 1 MB (it is not advisable to submit codes of size close to that or larger).
- The compilation time limit is 60 seconds. You can find information about compilers that we use, compilation options and processing server specifications here.

Examples

0)

```
Example Seed
```