Philadelphia Classic

The Dining Philosophers
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
dp-exec@googlegroups.com
http://dp.seas.upenn.edu

13 February 2010

1. Histograms and The Red-Headed League

In the original Sir Arthur Conan Doyle series, the story of The Red-Headed League involves an elaborate scheme to lore a feckless pawnbroker out of his shop which enables a criminal mastermind to use the location for his own devious purposes. In the process of creating The Red-Headed League, the story's antagonist interviews dozens of applicants and records various data about them including height information.

You are tasked with creating a program that accepts a comma-delimited list of heights and produces a histogram that will help to visualize the height distribution of applicants. The histogram should display in 1 inch increments from the shortest to the tallest height in the data. It is safe to assume that no person in the data set will be taller than 7'0". The histogram should display one asterisk symbol for each recorded height, and the columns should be labelled vertically with the corresponding height. The data should be separated from the labels on the x-axis by a row of dashes. All columns should have one space between them and a neighboring column. Finally, please note that the input may or may not be sorted.

Specification:
Implement the class makeHistogram, which consists of the method

```
public static String makeHistogram (String[] heights);
```

Example
Input:

```
makeHistogram("5’4\"","5’6\"","5’6\"","5’6\"","5’9\"","5’9\"","5’11\""}
```

Output:

```
    *
    *     *
*   *     *   *
---------------
5 5 5 5 5 5 5 5
’ ’ ’ ’ ’ ’ ’ ’
4 5 6 7 8 9 1 1
" " " " " " 0 1
            " "
```

2. The Case of The Misplaced Shooter

Sherlock Holmes is called to investigate the ballistics of a crime scene. The crime: murder. A single gunshot was fired in a closed room, at close range. However, the exact location of where the shooter was when the bullet was fired is still unknown. Your job is to deduce the exact location in the room that the bullet had when it left the barrel of the gun.

You have the following evidence: You know the room's exact dimensions. You know that the bullet went through the victim with trajectory unaltered. You know the muzzle velocity of the bullet. You know that when the bullet hit the victim, it lost 10% of it's initial speed, and it was the first thing the bullet hit. You know every time the bullet hit a wall, it deflected perfectly. You know that for every meter the bullet traveled in the room, it lost 1m/s of speed (after it hits the victim). You know the bullet came to rest inside of a wooden door at the entrance of the room. You know the position in the door the bullet was at, and the entrance vector of the bullet. You know that for every centimeter the bullet burrowed into the door, the bullet lost 50m/s of speed. You know the bullet could have hit the door before coming to final rest, but because it was going so fast, it deflected. Elementary. Good Luck!

```
10m separation between the walls
<-------------------------------->



             (0,0,0)
                .




<------------------------------->
```

The room is modeled as two infinite planes 10m apart from one another. The reference point is (0,0,0) and is exactly 5m distance from each plane. The +x axis points at the right, the +y axis points up, and the +z axis points at the far wall.

Specification: Implement the class Shooter, which consists of the method

```
public static double[] findShooterLocation (double muzzleVelocity,
    double holeX, double holeY, double holeZ,
    double entranceX, double entrancy Y, double entranceZ,
    double depth);
```

Example

Muzzle Velocity, Bullet Hole (x,y,z), Entrance Vector (x,y,z), Depth
Input: `findShooterLocation(100,0,0,-5,0,0,-1,1.5)`
Output: `[0.0 0.0 0.0]`

Input: `findShooterLocation(100,1,1,-5,0,0,-1,1.0)`
Output: `[1.0 1.0 -5.0]`

Input: `findShooterLocation(500,.25,-4,-5,1,-1,-1,1.5)`
Output: `[-216.25635094610868 212.5063509461087 -1.5063509461096838]`

Note: Precision of answers is at the judge's discretion. (Usually up to 3 or 4 decimal points)

3. Caesar's Challenge

As a renowned detective, Sherlock Holmes receives a plethora of fan mail. Some of his mail includes puzzles from would-be rivals, attempting to be the first to stump the master of deduction. Unfortunately, many of these correspondents are not very creative. In particular, he frequently gets "secret" messages encoded with Caesar ciphers. Although these ciphers are trivial for Holmes to crack, he has much better things to do, and has therefore passed this task on to Dr. Watson. Help relieve Watson of this boring job by writing a program to automatically decipher these messages.

A Caesar cipher takes as input a message string and a shift value n, and shifts each letter in the message forward through the alphabet by n letters. For example, shifting the letter 'A' forward by 1 would return 'B', and shifting 'B' forward by 2 would return 'D'. The alphabet should wrap around - shifting 'Z' forward by 1 would return 'A.' For example, shifting the word "HAL" forward by 1 would return "IBM", and shifting "Abcdefghijklm, nopqrstuvwxyz!" forward by 5 will return "Fghijklmnopqr, stuvwxyzabcde!". (Notice that case of letters is preserved, and non-letter symbols are unchanged.) Decoding a message encoded with shift n is equivalent to encoding a message with shift $26 - n$.

Of course, these rival-wannabes haven't given Holmes the shift value - your program has to figure it out itself. One way to do this is by shifting the message by all 26 possible shifts, and returning the one that seems most like English. To do this, you might find helpful this 26-element array, holding the frequencies of each letter in English:

```
double[] englishFrequencies =      {0.08167, 0.01492, 0.02782,
0.04253, 0.12702, 0.02228, 0.02015, 0.06094, 0.06966, 0.00153,
0.00772, 0.04025, 0.02406, 0.06749, 0.07507, 0.01929, 0.00095,
0.05987, 0.06327, 0.09056, 0.02758, 0.00978, 0.02360, 0.00150,
0.01974, 0.00074};
```

That is, 8.167% of all letters in English text are A, 1.492% are B, ... and 0.074% are Z.

Specification:
Implement the class CaesarDecipher, which consists of the method

```
public static String decipher (String ciphertext);
```

Example:

Input:

```
decipher("Me m dqzaizqp pqfqofuhq, Etqdxaow Taxyqe dqoquhqe m bxqftadm ar rmz ymux.")
```

Output:

```
As a renowned detective, Sherlock Holmes receives a plethora of fan mail.
```

4. BladeDodger

After solving the latest mystery, Sherlock Holmes decides to head home. He hops in a cab, gives the driver his address, and nods off as the driver starts winding his way through the London streets.

A few hours later Holmes awakes and realizes that he's been kidnapped and imprisoned in a holding cell with a ball and chain around his right ankle. The only way out is through a garbage chute lined with spinning blades. Holmes reasons that if he can make it down the chute, he can regain his freedom by finding a way out of the sewage system. He calls on you to work out the details.

BladeDodger
Provided: Blade.java BladeDodger.java
Submit BladeDodgerImpl.java

To describe the chute, Holmes will give you a `Blade[]` array. Each blade in the array can tell you its distance in meters from the top of the chute (`Blade[i].getDistance()`) and the times at which it is safe to pass by that blade ($n \times$ `Blade[i].getInterval()` where $n \geq 0$).

For example, the blade (distance 1, interval 2) is one meter from the top of the chute and is safe to pass by at times $t = 0, 2, 4, \ldots$. If Holmes starts sliding down the chute at any time $t = 1, 3, 5, \ldots$ he will safely pass by this blade.

Once he starts sliding, Sherlock Holmes will travel at a constant speed of one meter per second. Your task is to find the earliest time $t \geq 0$ (in seconds) at which Holmes can safely start his descent down the chute.

For example, if the blades are (distance 1, interval 2) and (distance 3, interval 5) then Holmes can safely start sliding at time $t = 7$. When he reaches the first blade, $t = 8 = n \times 2$ for $n = 4$ so he passes by safely. When he reaches the second blade, $t = 10 = n \times 5$ for $n = 2$ so again he passes safely.

You should read all the provided files before starting this question.

5. SewageSystemSolver

Provided SewageSystemSolver.java SewageSystem.java, SampleSewageSystems.java Submit SewageSystemImpl.java

Once he makes it into the sewage system, Holmes will need to find his way above ground. To describe his situation, he will give you a `SewageSystem` object called `maze` that provides Holmes's location (`maze.getX()`, `maze.getY()`) and the location of the maze exit (`maze.getExitX()`, `maze.getExitY()`). It also provides his direction `maze.getDirection()`. Holmes starts at (0, 0) facing `maze.SOUTH`.

The `maze` also describes the unfortunate restrictions Holmes faces in the sewage system. Thanks to the ball and chain around his right foot, Holmes can only `maze.turnRight()`. Thanks to the darkness, he must attempt to `maze.move()` in order to find out if there's a wall in front of him. (If there is a wall in front of him, `maze.move()` returns `false` and Holmes's coordinates are unchanged. If there's no wall, `maze.move()` returns `true` and Holmes moves forward.)

To get Holmes out of the sewage system, implement the right-hand rule algorithm. This strategy says that if you're in certain kinds of mazes, you can find the exit by keeping your right hand on the wall of the maze as you walk along.

For example, consider the following start state. (When you print the mazes, Sherlock is represented by a letter indicating his direction, the exit is represented as X, and the walls are represented by @ symbols.)

Example state:

```
S - - - - - - - - - - 
- - - - - - - - - - - 
- - - - - - - - - - - 
- - - - - - - - - - - 
- - - - - - - - - - - 
@ @ @ @ - - - - - - - 
- - - - - - - - - - - 
- - - - - - - - - - - 
- - - - - - - - - - - 
- - - - - - - - - - - 
- - - - - - - - - - X
```

The right-hand rule path for this maze is numbered below:

```
0 - - - - - - - - - - 
1 - - - - - - - - - - 
2 - - - - - - - - - - 
3 - - - - - - - - - - 
4 5 6 7 8 - - - - - - 
@ @ @ @ 9 - - - - - - 
F E C B A - - - - - - 
G - - - - - - - - - - 
H - - - - - - - - - - 
I - - - - - - - - - - 
J K L M N O P Q R S T
```

To help you develop your solver, we have provided four sample sewage systems.
Call `SampleSewageSystems.getSample(i)` where $1 \leq i \leq 4$ to retrieve the `SewageSystem` example number i.

6. Knights Tour

Detective Sherlock Holmes, in an effort to recover from a caseless week, seeks to stimulate his mind with the game of chess. However, Holmes mastered the game of chess years earlier, and found that a standard chess game was insufficient to stay his boredom. Instead, he turned his focus to the irregular motion of the Knight, and began determining a full walk around the board was feasible with a single knight.

In the interest of assisting Holmes, implement a function that determines, given an m by n chessboard and an initial position x,y of a knight on the board, whether or not the Knight can traverse the entire board without touching each square more than once.

Specification: Implement the class Knights, which consists of the method

```
public static boolean KnightsTour(int height, int widthm int startX, int startY);
```

Examples

Input: `KnightsTour(8,8,0,0)`
Output: `true`

Input: `KnightsTour(3,4,1,2)`
Output: `false`

7. Nim

A case has arrived and Holmes is in rare form. He and Watson grab the nearest cabbie, and they rush to the crimescene. On the way, Holmes and Watson debate who should have to pay the cabbie for his services. To settle the issue, they decided to play a game of Nim. The rules are as follows. The game starts with a set of piles of coins of varying numbers. On a player's turn he must remove one or more coins from any one pile. Whoever takes the last coin loses.

Holmes is not a man to be outsmarted, so you must devise a strategy to ensure he wins. Write a function that reads in an array of integers, decreases the value of a single entry in the array, and outputs a modified version of that array. The task of modifying the array is a trivial one, so the goal here is to think of a way to defeat a reasonable opponent. Your function will be competing with a simple Nim bot, so your goal should be to determine the best move possible.

Specification: Implement the class Nim, which consists of the method

```
public static int[] Nim (int[] piles);
```

Examples

Input: `Nim([1,7,3,1,2])`
Output: `[1,1,3,1,2]`

Input: `Nim([12,8,3,9,5,2])`
Output: `[12,8,3,0,5,2]`

8. The Problem of Making Change

Upon pursuing Prof. Moriarti by train, ship and motorized bicycle, Sherlock Holmes and Dr. Watson find themselves in the fictional kingdom of Classica, ruled by a temperamental gang of mathematicians who loathe visitors. Forced to pass themselves off as Classicians, Holmes and Watson are forced to obey the strange and cruel code of Classica convention.

Specifically, Classicians demand that whenever anything is purchased in Classica, exact change be paid, with as few bills and coins used as possible during the transaction. Supposedly for the mental exercise (but really out of spite), the Classician currency denomination system changes on a weekly basis. In order to blend in, Holmes and Watson will need to know how to make perfect change for an arbitrary amount of money in an arbitrary currency denomination system. Clearly, they need your help.

Specification:
Implement the class changeMaker, which consists of the method

```
public static int[] makeChange (int[] denominations, int amount);
```

1. denominations holds the various denominations (in ClassicaCredits) of the currencies available, in order. For example, the US system would be described by [1, 5, 10, 25, 100, 500... etc].
2. amount refers to the total amount for which Sherlock and John must make change.
3. the method returns an array of frequencies of the use of each type of denomination, in the same order as the input in denominations.

Examples

Input: `makeChange([1, 5, 10, 25, 100, 500], 456)`
Output: `[1, 1, 0, 2, 4, 0] (4 dollars, 2 quarters, a nickel and a penny) or [1, 1, 0, 2, 4] (see below)`

Input: `makeChange([1,2,4,8,16,32,64], 86)`
Output: `[0, 1, 1, 0, 1, 0, 1]`

Input: `makeChange([2,3,5], 6)`
Output: `[0,2,0]`

Input: `makeChange([1,5,6], 10)`
Output: `[0,2,0]`